IDC DOCUMENTATION

Event Magnitude Software

Approved for public release;
distribution unlimited

Notice
This document was published September 2001 by the Monitoring Systems Operation of Science Applications International Corporation (SAIC) as part of the International Data Centre (IDC) Documentation. Every effort was made to ensure that the information in this document was accurate at the time of publication. However, information is subject to change.

Contributors
Douglas A. Brumbaugh, Science Applications International Corporation

Ordering Information
The ordering number for this document is SAIC-01/3011.

This document is cited within other IDC documents as [IDC7.1.6].

Event Magnitude Software

CONTENTS

Event Magnitude Software

FIGURES

Event Magnitude Software

TABLES

About this Document

This chapter describes the organization and content of the document and includes the following topics:

- Purpose
- Scope
- Audience
- Related Information
- Using this Document

About this Document

PURPOSE

This document describes the design of the Event Magnitude (*libmagnitude*) software library and, to a lesser extent, the Event Location (*EvLoc*) software, which are elements of the International Data Centre (IDC). Both pieces of software are computer software components (CSCs) of the Automatic Processing computer software configuration item (CSCI). This document provides a basis for implementing, supporting, and testing both pieces of software.

SCOPE

This document describes the architectural and detailed design of the *libmagnitude* software library including its functionality, data structures, high-level interfaces, and methods of execution. This document also describes the architectural and detailed design of the *EvLoc* software necessary for magnitude estimation. This information is modeled on the Data Item Description for *Software Design Description* [DOD94a].

AUDIENCE

This document is intended for all engineering and management staff concerned with the design of all IDC software in general and of *libmagnitude* and *EvLoc* in particular. The detailed descriptions are intended for programmers who will be developing, testing, or maintaining *libmagnitude* and the magnitude functionality within *EvLoc*.

RELATED INFORMATION

The following documents complement this document:

- *Database Schema, Revision 2* [IDC5.1.1Rev2]
- *Event Location (libloc) Software* [IDC7.1.5]
- *IDC Processing of Seismic, Hydroacoustic, and Infrasonic Data* [IDC5.2.1]

See "References" on page 143 for a list of documents that supplement this document. The following UNIX manual (man) pages apply to the existing *libmagnitude* software:

- *libmagnitude*
- *EvLoc*

USING THIS DOCUMENT

This document is part of the overall documentation architecture for the IDC. It is part of the Software category, which describes the design of the software. This document is organized as follows:

- Chapter 1: Overview

  This chapter provides a high-level view of *libmagnitude* and *EvLoc*, including their functionality, components, background, status of development, and current operating environment.

- Chapter 2: Architectural Design

  This chapter describes the architectural design of *libmagnitude* and *EvLoc*, including their conceptual design, design decisions, functions, and interface design.

- Chapter 3: Detailed Design

  This chapter describes the detailed design of *libmagnitude* and *EvLoc*, including their data flow, software units, and database design.

- References

  This section lists the sources cited in this document.

- Glossary

  This section defines the terms, abbreviations, and acronyms used in this document.

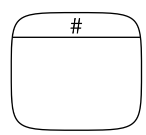
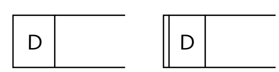
- Index

  This section lists topics and features provided in the document along with page numbers for reference.

Conventions

This document uses a variety of conventions, which are described in the following tables. Table I shows the conventions for data-flow diagrams. Table II shows the conventions for entity-relationship (E-R) diagrams. Table III lists typographical conventions.

TABLE I: DATA-FLOW SYMBOLS

Description	Symbol[1]
process	
data store (left), duplicated data store (right) M = memory store D = disk store Db = database store	
control flow	- - - - -▶
data flow	———▶

1. Most symbols in this table are based on Gane-Sarson conventions [Gan79].

TABLE II: ENTITY-RELATIONSHIP SYMBOLS

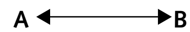
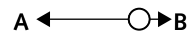
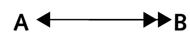
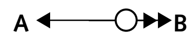
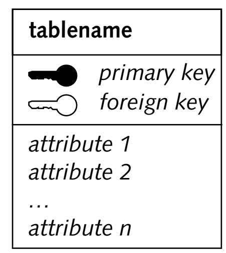
Description	Symbol
One **A** maps to one **B**.	
One **A** maps to zero or one **B.**	
One **A** maps to many **B**s.	
One **A** maps to zero or many **B**s.	
database table	

TABLE III: TYPOGRAPHICAL CONVENTIONS

Element	Font	Example
database table	**bold**	**stamag**
database table and attribute, when written in the dot notation		**netmag.***magnitude*
database attributes	*italics*	*uncertainty*
processes, software units, and libraries		*libmagnitude*
processing units		*setup_mag_facilities()*
variable names		*list_of_magtypes*
variables in output computer code		`Setting network sigma =` *<SGLIM1>*
titles of documents		*Software Design Description*
computer code and output	`courier`	`MDreadErr1: Cannot open MDF!`
filenames, directories, and websites		`mag_access.c`
text that should be typed in exactly as shown		`EvLoc par = EvLoc.par`
memory store component		`Mag_Params`
structure or object names		`Magnitude`
linked lists		`Ev`
data types		`char`

Chapter 1: Overview

This chapter provides a general overview of the *libmagnitude* and *EvLoc* software and includes the following topics:

- Introduction
- Functionality
- Identification
- Status of Development
- Background and History
- Operating Environment

Chapter 1: Overview

INTRODUCTION

The software of the IDC acquires time-series and radionuclide data from stations of the International Monitoring System (IMS) and other locations. These data are passed through a number of automatic and interactive analysis stages, which culminate in the estimation of the location and origin time of events (earthquakes, volcanic eruptions, and so on) in the earth, including its oceans and atmosphere. The results of the analysis are distributed to States Parties and other users by various means. Approximately one million lines of developmental software are spread across six computer software configuration items (CSCIs) of the software architecture. One additional CSCI is devoted to run-time data of the software. Figure 1 shows the logical organization of the IDC software. The Automatic Processing CSCI processes data through the following computer software components (CSCs):

- Station Processing

  This software scans data from individual time-series stations for characteristic changes in the waveforms (detection of onsets) and characterizes such onsets (feature extraction). The software then classifies the detections as arrivals in terms of phase type.

- Network Processing

  This software combines arrivals from several stations originating from one event and infers the location and time of its origin.

- Post-location Processing

  This software computes various magnitude estimates and selects data to be retrieved from auxiliary stations.

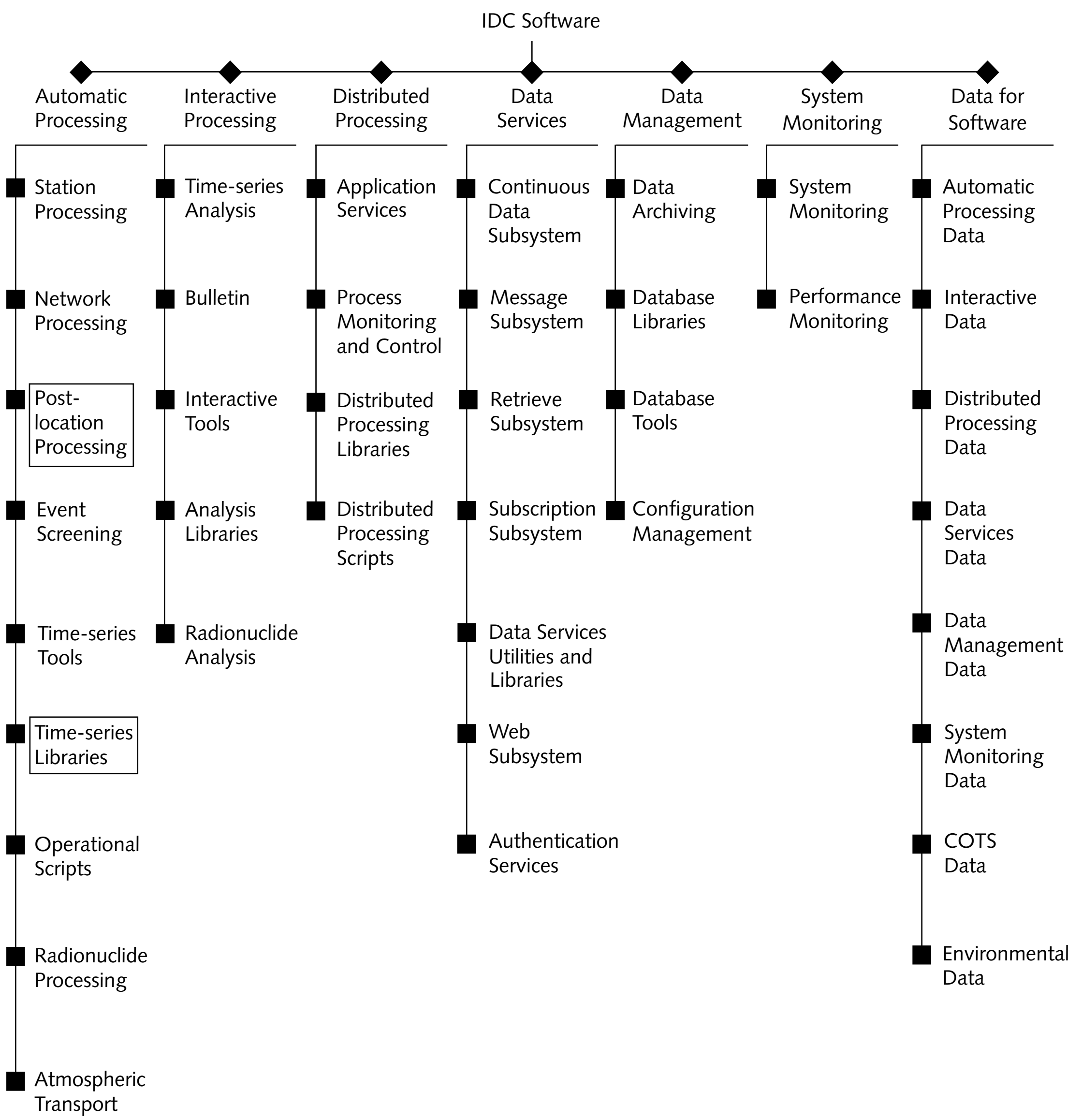


FIGURE 1. IDC SOFTWARE CONFIGURATION HIERARCHY

- Event Screening

  This software extracts a number of parameters that characterize an event; then a default subset of the calculated Event Characterization Parameters eliminates the events that are clearly not explosions.

- Time-series Tools

  This software includes various utilities for the Seismic, Hydroacoustic, and Infrasonic (S/H/I) processing system.

- Time-series Libraries

  This software includes shared libraries to which several modules of the S/H/I processing system are linked.

- Operational Scripts

  This software provides miscellaneous functionality to enable Automatic Processing to function as a system.

- Radionuclide Processing

  This software includes the automated analysis, categorization, and flagging processes for radionuclide data.

- Atmospheric Transport

  This software includes the forward and backward modeling of the transport of particulates by atmospheric movements.

The *libmagnitude* common software library resides in the Time-series Libraries CSC of the Automatic Processing CSCI. However, this library is used in multiple applications in the Automatic Processing and Interactive Processing CSCIs. The *libmagnitude* library is used by the Station Processing (*StaPro*) application in the Station Processing CSC, the Global Association (GA) Subsystem in the Network Processing CSC, and the *WaveExpert* and Event Location (*EvLoc*) applications in the Post-location Processing CSC of the Automatic Processing CSCI. The *libmagnitude* library is also used by the Analyst Review Station (*ARS*) application in the Time-series Analysis CSC of the Interactive Processing CSCI.

Figure 2 shows the relationship of *libmagnitude* to the applications of the Automatic Processing and Interactive Processing CSCIs mentioned in the previous paragraph. At the request of a user (that is, the operator or Distributed Applications Control System [DACS] pipeline process), which is not shown in Figure 2, these applications read station and event data from an input database account and control parameters from parameter/Scheme files. They pass station and event data and magnitude-related control parameters to one or more *libmagnitude* interfaces, which distribute control and data to other *libmagnitude* processes. The *libmagnitude* processes acquire data from a set of earth-model files. After the requested processing is complete, the resulting magnitude data are returned to the applications. The applications in Figure 2 write updated event data to an output database. These event data may or may not include the magnitude information returned from *libmagnitude*.

This document mainly describes the design elements of the *libmagnitude* software. However, because *libmagnitude* is a library, a description of how applications interface with it is essential. *EvLoc* calls most of the *libmagnitude* processing units and therefore serves as a convenient application for describing these interfaces. For completeness, this document also describes the design elements of *EvLoc*.

FUNCTIONALITY

The *libmagnitude* software has two basic modes of operation: estimating station-magnitude data and estimating network-magnitude data. In the station-magnitude mode, magnitude corrections are applied to event data to estimate station-magnitude data. The magnitude corrections are read from a set of earth-model files (D2 in Figure 2). In the network-magnitude mode, network-magnitude data are estimated from station-magnitude data. Because station-magnitude data are necessary to estimate network-magnitude data, the computation of station-magnitude data is a necessary process within the network-magnitude mode. These two modes are described in more detail in “Conceptual Design” on page 10.

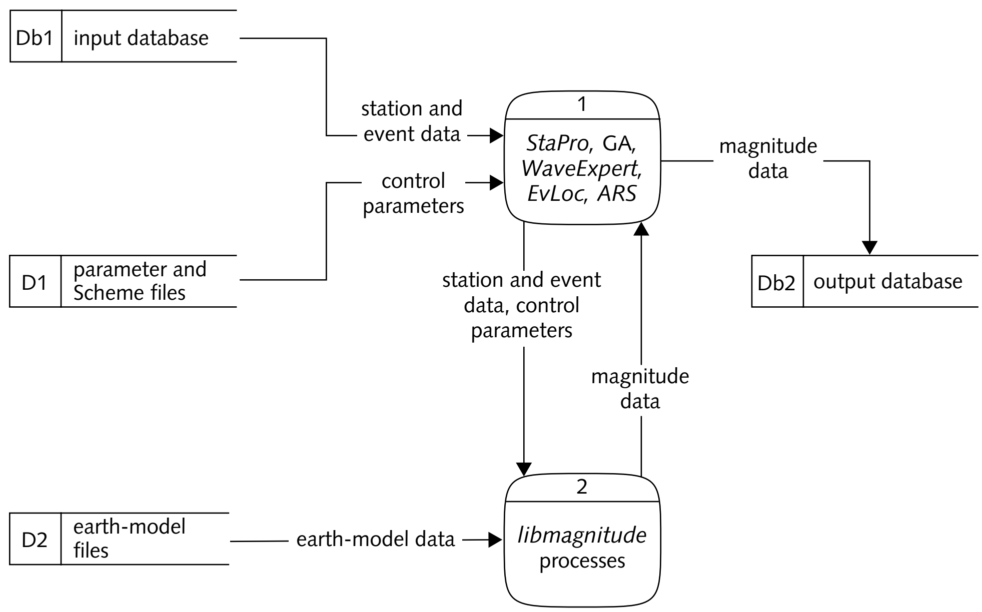


FIGURE 2. RELATIONSHIP OF LIBMAGNITUDE TO OTHER SOFTWARE UNITS

A user may not directly call any *libmagnitude* functions. The user must use an application to access the functionality provided by *libmagnitude*. Figure 2 indicates five applications that access *libmagnitude*. *StaPro*, the GA Subsystem, and *WaveExpert* only operate in the station-magnitude mode. *EvLoc* and *ARS* operate in the network-magnitude mode.

The *EvLoc* software estimates event location and magnitude data either in combination or as independent processes. Although magnitude data may not be estimated without first locating events, all discussion of *EvLoc* in this document assumes that events have already been located and that *EvLoc* is processing magnitude data only. Refer to [IDC7.1.5] for information about how *EvLoc* processes event location data.

IDENTIFICATION

The event-magnitude software components are identified as follows:

- *libmagnitude*
- *EvLoc*

STATUS OF DEVELOPMENT

The *libmagnitude* and *EvLoc* software is mature and stable. This software is undergoing limited enhancements on a continuing basis.

BACKGROUND AND HISTORY

Keith McLaughlin, Hans Israelsson, Steve Bratt, Walter Nagy, Jeffrey Given, and SAIC staff began developing *libmagnitude* in 1989. At the time, this magnitude software library was known as *libmagn* instead of *libmagnitude*. The original *libmagn* software was written in FORTRAN. In 1997, Walter Nagy of SAIC converted the FORTRAN source code to C and renamed the library.

libmagn was first used operationally in 1989 as part of the GSETT-3 effort. *libmagnitude* was first used operationally in 1998. *libmagnitude* currently is an element of the Prototype IDC (PIDC) at the Center for Monitoring Research (CMR) in Arlington, Virginia, USA and at the International Data Centre of the Comprehensive Nuclear-Test-Ban Treaty Organization (CTBTO IDC) in Vienna, Austria. *EvLoc* is also used both operationally and as a research tool in a number of other systems.

OPERATING ENVIRONMENT

The following paragraphs describe the hardware and commercial-off-the-shelf (COTS) software required to operate *EvLoc* and *libmagnitude*.

Hardware

EvLoc is designed to run on a UNIX workstation, such as the SPARC-20 manufactured by Sun Microsystems. Typically, the hardware is configured with 256 MB of memory and a minimum of 2 GB of magnetic disk, although *EvLoc* could be executed on systems with smaller resources, depending on the number and type of other resident processes. The required memory will scale roughly with the number of events processed. *EvLoc* obtains database access over an Ethernet connection to other computers residing on a Local Area Network. Figure 3 shows a representative hardware configuration. Using this hardware configuration, *EvLoc* typically requires less than 5 seconds of processing time to estimate station magnitudes and a network-average magnitude for all events in an hour-long segment of analyst-reviewed seismic data.

libmagnitude does not require any special hardware configuration.

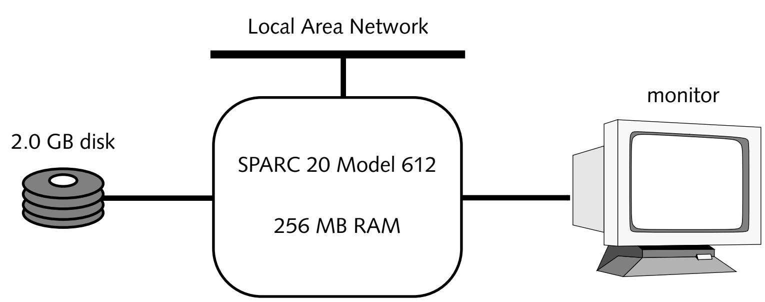


FIGURE 3. REPRESENTATIVE HARDWARE CONFIGURATION FOR EVLOC

Commercial-Off-The-Shelf Software

The *libmagnitude* and *EvLoc* software are designed for Solaris 2.7 and ORACLE 8.1.5. No other COTS software is required.

Chapter 2: Architectural Design

This chapter describes the architectural design of *libmagnitude* and *EvLoc* and includes the following topics:

- Conceptual Design
- Design Decisions
- EvLoc Functional Description
- libmagnitude Functional Description
- EvLoc Interface Design
- libmagnitude Interface Design

Chapter 2: Architectural Design

CONCEPTUAL DESIGN

The purpose of the *EvLoc* software is to acquire data from input data stores (database and static data files), exchange it with the *libmagnitude* and *libloc* software libraries, and write the resulting data to an output data store (database). These two software libraries perform most of the processing; *EvLoc* is primarily a medium between the data storage units and the critical processing units. *EvLoc* reads control parameters from an input parameter file, and station and event data from the database, and passes these data to one or both libraries for determining magnitude/location estimates. The resulting magnitude/location data are returned to *EvLoc*, which writes the results to the database.

The purpose of the *libmagnitude* software is to provide a set of software interfaces for reading and storing input event and earth-model data, estimating station-magnitude data, and estimating network-magnitude data. The software design assumes that input events have already been located by the *libloc* software. The functionality provided by the *libmagnitude* interfaces and their associated lower-level processing units comprises the two functional modes of operation discussed in "Functionality" on page 5. Specifically, the station-magnitude mode reads and stores input earth-model data and estimates station-magnitude data. The network-magnitude mode uses the functionality of the station-magnitude mode coupled with the input event data to estimate network-magnitude data.

One of the central *libmagnitude* elements is the `Magnitude` object (see Table 54 on page 133). *libmagnitude* was designed to be more object-oriented than its predecessor, *libmagn*, so the `Magnitude` object was established as the central (shared) data store. This `Magnitude` "object" is not a true object as often used in object-oriented programming in that it only contains data and not the functions applied to the data. Strictly speaking, the `Magnitude` object is a complex data structure. Multiple *libmagnitude* processing units and even external applications may access the data contained within this object, but only when they are operating in the network-magnitude mode of operation. A detailed description of the design and usage of the `Magnitude` object is provided in "build_mag_obj()" on page 88.

DESIGN DECISIONS

The following design decisions pertain to *EvLoc* and *libmagnitude*.

Programming Language

Each software unit of *EvLoc* and *libmagnitude* is written in the C programming language. This supports efficient processing and convenient integration with other components of the IDC system.

Global Libraries

The *EvLoc* software requires the following shared libraries: *libgdi*, *libloc*, *libmagnitude*, *libinterp*, *libgeog*, *libLP*, *libstdtime*, *libdb30qa*, *libpar*, and *libaesir*. The *EvLoc* software is also linked to the following COTS libraries: *libF77*, *libtermcap*, *libsocket*, *libnsl*, *libelf*, *libm*, *libdl*, and *libsunmath*.

The *libmagnitude* software requires the following shared libraries: *libinterp*, *libgeog*, *libstdtime*, and *libaesir*. The *libmagnitude* software requires the *libm* COTS library.

Database

EvLoc accesses an ORACLE database account and reads data from **site**, **origin**, **assoc**, **amplitude**, **parrival**, **stamag**, and **netmag** tables. *EvLoc* also requires data from an **affiliation** table to link the stations assigned to a network to data from the input **site** table. *EvLoc* may also use an **event_control** table to retrieve or set a number of magnitude control parameters. These data are stored internally in analogous database table structures, most of which are then passed to *libmagnitude* processes. Upon successful completion of *libmagnitude* processing, *EvLoc* writes data returned from *libmagnitude* to the output **origin**, **stamag**, **netmag**, and optionally

event_control database tables. Other applications, such as *ARS*, perform tasks similar to *EvLoc*, although the list of input or output database tables may differ. Descriptions of these tables are given in [IDC5.1.1Rev2].

Filesystem

EvLoc uses parameter files to specify program control. Other applications that use *libmagnitude*, such as *ARS*, also use Scheme files for this purpose. *libmagnitude* is a software library, so it does not read parameter or Scheme files. Instead, program control parameters are passed from *EvLoc* or any calling application to it.

libmagnitude requires input earth-model data. The earth-model data are stored on the filesystem in ASCII flat files and are read by *libmagnitude* processing units (Figure 2 on page 6).

In addition, the user may want to direct output from *EvLoc*/*libmagnitude* processing units to log files. At present, the UNIX filesystem is the only supported filesystem for which *libmagnitude* and *EvLoc* have been validated.

Design Model

The design of *libmagnitude* is primarily influenced by commonality, flexibility and extensibility, maintainability, and timeliness requirements. The commonality requirement is that all IDC applications estimate magnitudes in the same way. This is addressed by providing a set of standard interfaces that utilize common procedures and earth-model data that can be employed by all IDC applications. Flexibility and extensibility are necessary to allow future support for additional magnitude types. This is addressed by implementing a standard representation and application of transmission-loss data, which is applicable to all magnitude types. This information is managed by standard Transmission-Loss Specification File (TLSF) and Magnitude Description File (MDF). Maintainability is addressed by the use of these two specification files and the use of C as a standard implementation language. Timeliness is addressed by using C to efficiently access, store, and process the earth-model data.

Database Schema Overview

EvLoc uses the ORACLE database tables for the following purposes:

- to retrieve station data (**site**, **affiliation**)
- to retrieve event and signal data (**origin**, **assoc**, **parrival**, **amplitude**, **stamag**, **netmag**)
- to retrieve optional magnitude control parameters (**event_control**)
- to store updated magnitude information (**origin**, **netmag**, **stamag**, **event_control**)

Many *EvLoc* processing units require reading data from or writing data to database tables. The data retrieved from or written to the database tables are stored internally in C database table structures that are structurally equivalent to the database tables themselves. Table 1 shows the database tables used by *EvLoc*. The Name column identifies the database table. The Mode column is "R" if *EvLoc* reads from the table and "W" if *EvLoc* writes to the table.

TABLE 1: DATABASE TABLES USED BY EVLOC

Name	Mode	Description
affiliation	R	station network information
site	R	station location information
origin	R/W	origin information for particular event, including location and magnitude estimates
assoc	R	arrival association information
parrival	R	predicted arrivals and associations for origin-based amplitude measurements
amplitude	R	amplitude measurements
stamag	R/W	station-magnitude estimates
netmag	R/W	network-magnitude estimates
event_control	R/W	event location and magnitude control parameters (optional)

libmagnitude does not access the database tables directly; it only uses the database table structures provided by the calling application. Table 2 shows the database table structures used by *libmagnitude*. The Name column identifies the database table structure. The Mode column is "R" if *libmagnitude* uses a table structure read from an input database account and "W" if *libmagnitude* populates a table structure written to an output database account.

TABLE 2: DATABASE TABLE STRUCTURES USED BY LIBMAGNITUDE

Name	Mode	Description
`Site`	R	station location information
`Origin`	R/W	origin information for particular event, including location and magnitude estimates
`Assoc`	R	arrival association information
`Parrival`	R	predicted arrivals and associations for origin-based amplitude measurements
`Amplitude`	R	amplitude measurements
`Stamag`	R/W	station-magnitude estimates
`Netmag`	R/W	network-magnitude estimates

The `Affiliation` and `Event_Control` database table structures are not used by *libmagnitude*. *EvLoc* reads **affiliation** and **event_control** data and stores them in database table structures. However, *EvLoc* only uses the **affiliation** data to retrieve station location data from the input **site** table and *EvLoc* copies the magnitude control parameters from the **event_control** data into a different C structure that is passed to *libmagnitude*.

EVLOC FUNCTIONAL DESCRIPTION

The main processes of *EvLoc*, as they relate to magnitude processing, are described in this section. Because *libmagnitude* is a library, *EvLoc* is employed as a specific application vehicle to demonstrate how *libmagnitude* functions are commonly exploited. *EvLoc*, along with *ARS*, operates in network-magnitude mode and therefore invokes all of the primary *libmagnitude* interfaces. Other applications (see "Functionality" on page 5) operate in station-magnitude mode and consequently only invoke some of the primary *libmagnitude* interfaces.

Figure 4 shows the *EvLoc* functional design. In this and subsequent data flow diagrams, *EvLoc* processes are assigned unique identifiers of the form "1.*n*". *EvLoc* has four main processes. First, it reads control-parameter arguments from an input parameter file and optional magnitude control parameters from an input database. Second, *EvLoc* reads station data from the input database tables (Table 1). Third, it reads event data from the input database tables. The control parameters, station data, and event data are put in memory stores that are passed to *libmagnitude* for processing. The memory stores are identified and described in "EvLoc Data flow Model" on page 30. When *libmagnitude* processing is complete, any updated event results (station- and network-magnitude data) are returned to *EvLoc*. The fourth main process of *EvLoc* obtains the updated magnitude data and writes it to output database tables (Table 1).

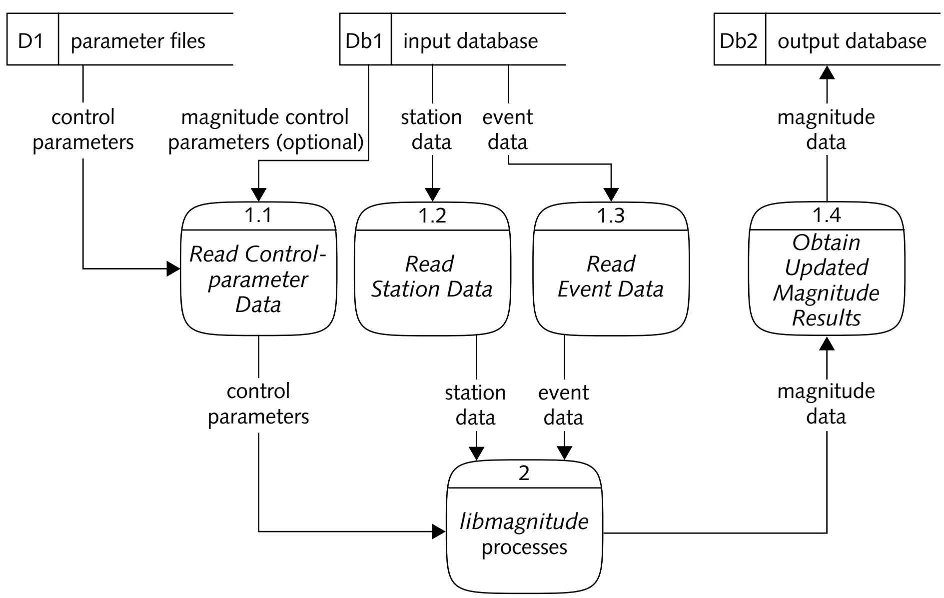


FIGURE 4. EVLOC FUNCTIONAL DESIGN

Read Control-parameter Data

The first main *EvLoc* process (1.1 in Figure 4) reads and stores control parameters that are necessary for magnitude processing. Most control parameters are read from an input parameter file (D1), but some may optionally be read from an input database account (Db1).

Process 1.1 reads control parameters from the parameter file using interfaces from the *libpar* library. These parameters are either required or optional. Required parameters must be specified or *EvLoc* immediately exits and returns an error message. Optional parameters have default states that are overridden only if specified in the input parameter file. The parameters define general and magnitude-specific information such as input and output database accounts and tables, station network, events to be processed, directory locations of earth-model files, and additional magnitude control parameters. These control-parameter data are stored in memory. (Refer to the *EvLoc* man page for descriptions of available parameters.)

Process 1.1 may optionally read additional magnitude control parameters from the **event_control** table (see Table 1 on page 13) using interfaces from the *libgdi* library. The **event_control** table contains control information that may vary on an event-by-event basis. These magnitude control parameter data are also stored in memory.

Read Station Data

The second main *EvLoc* process (1.2 in Figure 4) reads station data from an input database account (Db1) and stores them in memory. Process 1.2 reads the station data from the **site** and **affiliation** tables (see Table 1) using interfaces from the *libgdi* library. The station-network relationship within the **affiliation** table defines which subset of station records are to be retrieved from the **site** table. This process copies the **site** records into `Site` database table structures (see Table 2 on page 14) and stores them in memory. The `Site` structures are required by the *libmagnitude* interface for reading earth-model files.

Read Event Data

The third main *EvLoc* process (1.3 in Figure 4) reads event data from an input database account (Db1) and stores them in memory. Process 1.3 reads the event data from the **origin**, **assoc**, **amplitude**, **parrival**, **stamag**, and **netmag** tables (see Table 1) using interfaces from the *libgdi* library. It copies records from these tables into `Origin`, `Assoc`, `Amplitude`, `Parrival`, `Stamag`, and `Netmag` database table structures (see Table 2) and stores them in memory. These structures are required by several *libmagnitude* interfaces.

Obtain Updated Magnitude Results

The fourth main *EvLoc* process (1.4 in Figure 4) obtains updated station- and network-magnitude data estimated by *libmagnitude* and writes these results to an output database account (Db2). Process 1.4 writes the updated magnitude data to the **origin**, **stamag**, **netmag**, and optionally, **event_control** tables (see Table 1) using interfaces from the *libgdi* library. The **origin**, **stamag**, and **netmag** records are created by this process from `Origin`, `Stamag`, and `Netmag` database table structures (see Table 2) passed from *libmagnitude*. If specified by control parameters, **event_control** records are created by this *EvLoc* process and written to the database.

Process 1.4 may also be configured to output log file results to the filesystem in a simple ASCII format.

LIBMAGNITUDE FUNCTIONAL DESCRIPTION

The main processes of *libmagnitude* are described in this section.

libmagnitude has four main processes. However, the processes that are accessed depend on the mode of operation. When an application operates in station-magnitude mode, only two main *libmagnitude* processes are accessed. When an application operates in network-magnitude mode, all four main processes are accessed.

Figure 5 shows the *libmagnitude* functional design in station-magnitude mode. In this and subsequent data flow diagrams, *libmagnitude* processes are assigned unique identifiers of the form "2.*n*". Two main *libmagnitude* processes are associated with this mode. First, *libmagnitude* reads a set of earth-model files (2.1). Second, it estimates station-magnitude data from event data (primarily amplitude data) and earth-model data (2.3). In Figure 5, this process is labeled 2.3 instead of 2.2 because another process is present between these two processes in the network-magnitude mode functional design.

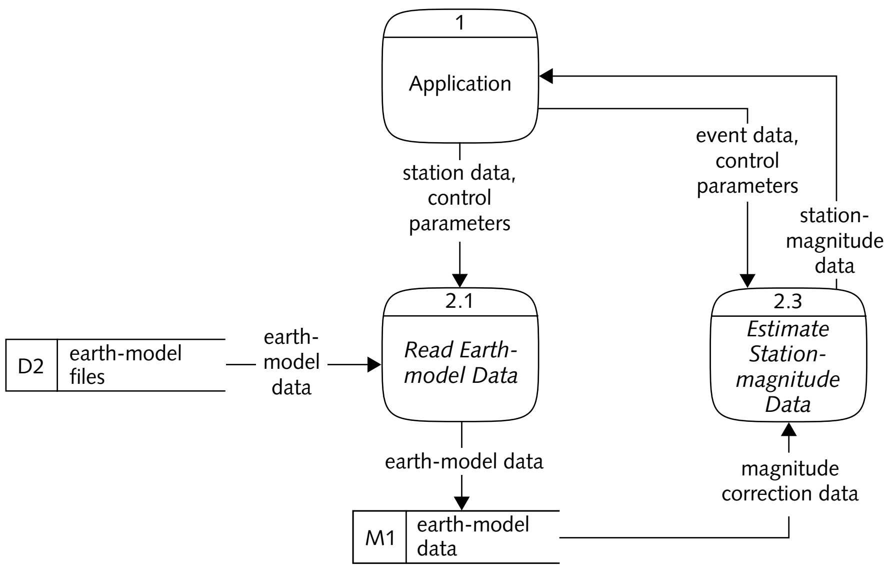


FIGURE 5. LIBMAGNITUDE FUNCTIONAL DESIGN IN STATION-MAGNITUDE MODE

Figure 6 shows the *libmagnitude* functional design in network-magnitude mode. All four main *libmagnitude* processes are associated with this mode. First, *libmagnitude* reads a set of earth-model files (2.1). Second, it stores event data and magnitude specification data in a memory store (2.2). Third, *libmagnitude* estimates station-magnitude data from event data (primarily amplitude data) and earth-model data (2.3). Fourth, it estimates network-magnitude data from station-magnitude data using up to four different computational algorithms (2.4).

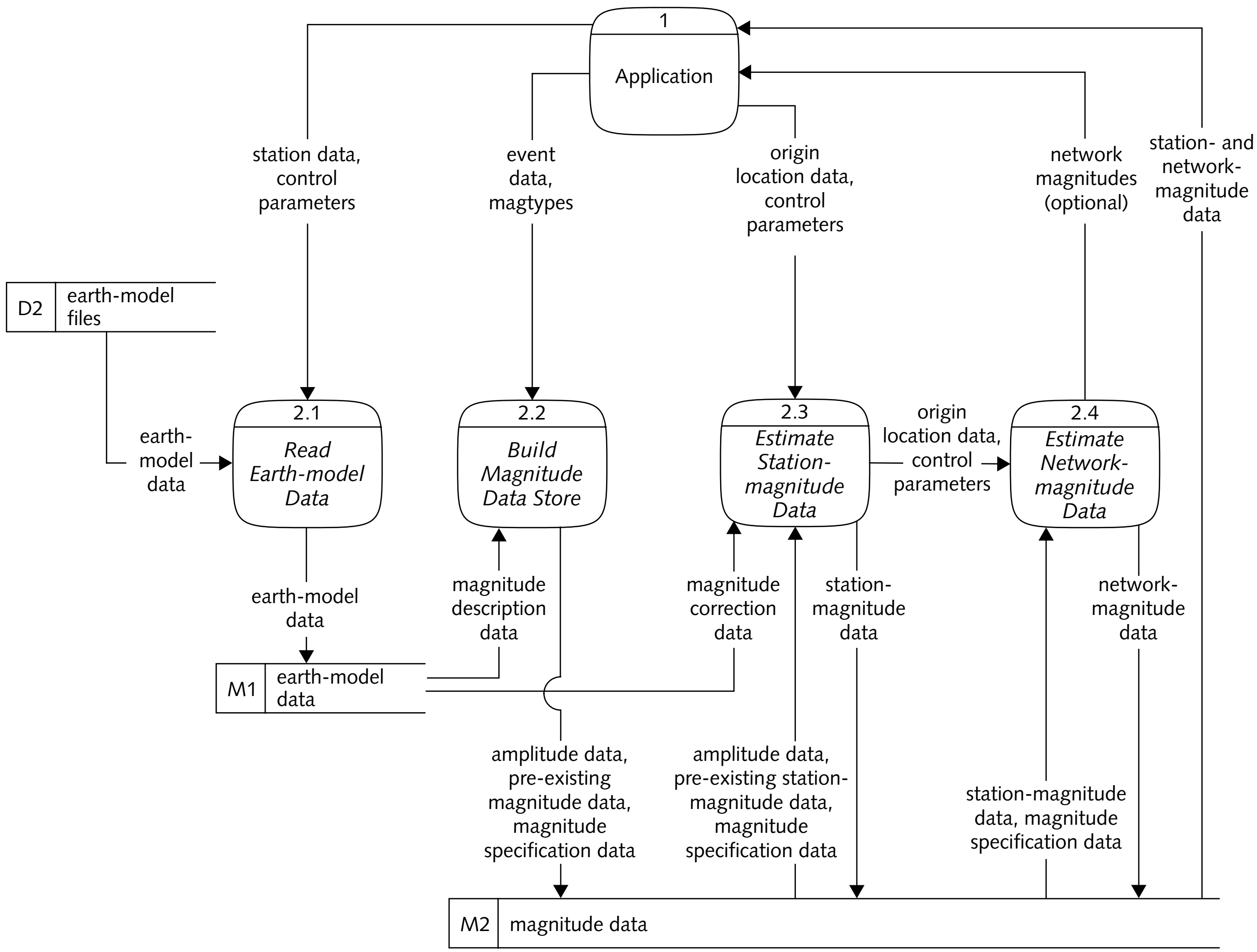


FIGURE 6. LIBMAGNITUDE FUNCTIONAL DESIGN IN NETWORK-MAGNITUDE MODE

libmagnitude processes may not be called directly by a user. However, any application may access any of the four primary processes. The only constraints are that the processes must be accessed in the order they are presented in Figures 5 and 6.

Read Earth-model Data

The first main process of *libmagnitude* reads a set of earth-model files (D2 in Figure 5 and Figure 6) and stores the contents in an earth-model-data memory store (M1 in Figures 5 and 6). This process (2.1 in Figures 5 and 6), is performed regardless of the operational mode.

The calling application passes a set of station data and control parameters to this *libmagnitude* process. These data define which earth-model data should be read from the earth-model files (D2) and stored in the earth-model-data memory store (M1). The earth-model files are a collection of three different types of ASCII flat files: a MDF, a TLSF, and one or more Transmission-Loss Models (TLMs). The MDF defines magnitude control settings for any number of magnitude descriptors (magtypes), links magtypes to transmission-loss descriptors (TLtypes), and specifies bulk-station-correction data given TLtype-station combinations. The TLSF is the central control point for defining all regionalized transmission-loss knowledge. The TLSF links TLtypes to default TLMs, links station-TLtype (and optionally phase and channel/frequency) combinations to station-specific TLMs, and associates phase names with TLtypes. This latter association ultimately defines the phases used to compute a given magtype through the magtype-TLtype link in the MDF. The TLMs contain distance/depth-dependent magnitude corrections and modeling errors. The formats and structures associated with these three file types are described in conjunction with the *libmagnitude* processing units that parse them in “read_mdf()” on page 77, “read_tlsf()” on page 80, and “read_tl_table()” on page 85.

The TLSF is designed to be the central control point connecting control parameters to raw transmission-loss correction data. It has the capacity to serve as a control point for other sets of control parameters that require access to raw data files as well. Presently, only magnitude computations use this particular design.

Build Magnitude Data Store

The second main process of *libmagnitude* stores input event and magnitude specification data in a magnitude-data memory store (M2 in Figure 6 on page 19). The data are stored in `Magnitude` objects (see Table 54 on page 133). This process (2.2 in Figure 6) is only performed in network-magnitude mode.

This process stores input event data passed from the calling application and magnitude description data read from the earth-model-data memory store (M1) for an input list of magtypes. The event data are the six database table structures listed in Table 2 on page 14 that contain event information. The amplitude data are copied into M2. If station- and network-magnitude data already exist for any of the input events, then these "pre-existing" data are also copied into M2. The magnitude description data are magnitude control settings stored in M1. The magnitude description data are copied into `Magnitude` objects, renamed as magnitude specification data, and stored in M2. Consolidation of data relevant to computing station- and network-magnitude data into one memory store permits easy access to these data.

Estimate Station-magnitude Data

The third main process of *libmagnitude* estimates a station magnitude, uncertainty, and other ancillary station-magnitude data. This process (2.3 in Figure 5 on page 18 and Figure 6), is performed regardless of the operational mode. However, the mode dictates where the input data are read from and where the output station-magnitude data are stored.

In station-magnitude mode (Figure 5) the calling application passes event data and control parameters to process 2.3. Process 2.3 then reads magnitude correction data in the form of distance/depth adjustments from M1 and applies them to input event (for example, amplitude and period) data to estimate a station magnitude. Process 2.3 also reads the modeling errors from M1 and incorporates them into the uncertainty estimation process. The resulting station magnitude, uncertainty, and other determined station-magnitude data are returned to the application.

In network-magnitude mode (Figure 6) the calling application also passes event data and control parameters to process 2.3. However, the event data are only origin location information and do not include amplitude data. The amplitude data, pre-existing magnitude data, and magnitude specification data are read from the magnitude-data memory store (M2) by process 2.3. This process then estimates station magnitudes, uncertainties, and other station-magnitude data (as described in the previous paragraph) using the magnitude corrections and modeling errors read from the earth-model-data memory store (M1). This process also uses the control parameters and magnitude specification data to identify which station magnitudes will be used to estimate network magnitudes. This additional functionality is not present in station-magnitude mode.

The resulting station-magnitude data are stored within the `Magnitude` objects (M2). The application may retrieve the station-magnitude data from M2 after process 2.3 successfully completes.

Estimate Network-magnitude Data

The fourth main process of *libmagnitude* estimates a network magnitude, uncertainty, and other ancillary network-magnitude data. This process (2.4 in Figure 6), is only performed in network-magnitude mode.

This process estimates network-magnitude data by using origin location data and control parameters passed from the calling application through process 1.3, and station-magnitude data and magnitude specification data read from the magnitude-data memory store (M2). Network m_b, M_s, and M_L magnitudes and uncertainties are estimated using any of the following algorithms: network average, weighted network average, maximum-likelihood estimate (MLE), weighted MLE, upper- or lower-magnitude bounds, and weighted upper- or lower-magnitude bounds. The MLE magnitude uncertainty may also be determined by a bootstrap resampling technique [McL88]. The control parameters and magnitude specification data define how these algorithms use the station-magnitude data to estimate network-magnitude data. Upper- or lower-magnitude bounds may only be estimated if all available amplitudes are theoretical or clipped, respectively.

The resulting network-magnitude data are stored within the `Magnitude` objects (M2). The application may retrieve the network-magnitude data from M2 after process 2.4 successfully completes.

EVLOC INTERFACE DESIGN

This section describes *EvLoc*'s interfaces with other IDC systems, external users, and operators.

Interface with Other IDC Systems

EvLoc is an important software component of the Post-location Processing pipeline in the IDC Automatic Processing CSCI. [IDC6.5.2Rev0.1] describes the initialization criteria, control flow, and configuration of this pipeline. Briefly, the Post-location Processing pipeline initiates after an analyst pushes the "delpass" or "scanpass" buttons in the *analyst_log* application of the Interactive Processing CSCI. A *tis_server* data monitor with the service name of *tis-recall* creates intervals and a succession of queues and *tuxshell*s that launch the applications constituting the Post-location Processing pipeline. The *tuxshell* processing server assembles a command line for each application or child process, submits the command line to the operating system, monitors its execution, and evaluates the exit status. If the child process is successful, *tuxshell* sends a message to the next processing queue to launch the next child process. The configuration of the Post-location Processing pipeline is described in Table 3.

TABLE 3: CONFIGURATION OF POST-LOCATION PROCESSING PIPELINE

Queue and tuxshell Service Name	Child Process Name	Processing Description
DFX-recall	*DFX*[1]	revises seismic waveform measurements after analyst review[2]
DFX-dphase-SNR	*DFX*[1]	computes specialized snr for event-associated depth phases[2]
DFX-noiseamp	*DFX*[1]	estimates amplitudes for theoretical arrivals at stations that did not detect arrivals from a given origin[2]
EvLoc-mb_ave	*EvLoc*	estimates m_b network averages
EvLoc-mb_mle	*EvLoc*	estimates m_b MLEs
EvLoc-mb1	*EvLoc*	estimates m_b weighted network averages and m_b weighted MLEs
EvLoc-mlppn	*EvLoc*	estimates M_L weighted network averages
MsOrid	*MsInterval, MsOrid, maxsurf*	tests for existence of surface waves, measures amplitudes, and estimates M_s network averages, M_s MLEs, M_s weighted network averages, and M_s weighted MLEs[3]

1. *Detection and Feature Extraction* application of the Automatic Processing CSCI.
2. See [IDC7.1.1].
3. The four M_s network magnitudes are estimated through two executions of *EvLoc* that are a part of the *MsOrid* child process.

During *EvLoc* execution, which is initiated either by *tuxshell* or *MsOrid*, *EvLoc* reads station and event data from the input database and writes magnitude data to the output database through *libgdi* interfaces (Figure 4 on page 15). The input event data are populated either by *ARS* or *DFX*. The event data from *ARS* are written to the database after review by an analyst. The event data from *DFX* are estimated during the *DFX-noiseamp* processing step (Table 3). *DFX* does not initiate *EvLoc* or interface with *libmagnitude*, so it is not discussed in further detail; a description of the *DFX* application design is available in [IDC7.1.1]. The output magnitude data are available in the output database at the completion of the Post-location Processing pipeline for use by applications in a later pipeline such as the Reviewed Event Bulletin (REB) pipeline [IDC6.5.2Rev0.1].

Interface with External Users

EvLoc may be executed directly from the command line by specifying the executable name and an input parameter file, such as:

```
EvLoc par=EvLoc.par
```

The ability to execute *EvLoc* outside of a pipeline allows researchers and testers to tune, test, and experiment with location and magnitude parameters on varying sets of input station and event data.

Interface with Operators

EvLoc writes error messages to standard error. In IDC operations, these messages are generally redirected to a log file. Such messages may provide clues to the operator if processing fails. *EvLoc* also writes informational messages to standard output. These messages may help to tune the configuration.

LIBMAGNITUDE INTERFACE DESIGN

This section describes *libmagnitude*'s interfaces with other IDC systems, external users, and operators.

Interface with Other IDC Systems

libmagnitude only interfaces with other IDC systems to the extent that it reads much of its required earth-model data from the filesystem. However, applications that use *libmagnitude* interfaces do interact with other IDC systems through database reads and writes and through filesystem access.

Figure 2 on page 6 shows that *StaPro*, the GA Subsystem, *WaveExpert*, *EvLoc*, and *ARS* utilize *libmagnitude* processes. *StaPro*, the GA Subsystem, and *WaveExpert* use *libmagnitude* in station-magnitude mode to read earth-model data and estimate station-magnitude data. *StaPro* writes the station magnitudes to the **origin** table, but does not write **netmag** or **stamag**. The GA Subsystem uses the station-magnitude data to estimate network-magnitude data independent of *libmagnitude*. The GA grid constructor (*GAcons*), which is a component of the GA Subsystem, writes a set of magnitude correction derivatives that are computed in conjunction with station magnitudes to a binary GA grid file. *WaveExpert* uses the distance-depth magnitude corrections and modeling errors read from the TLMs to estimate the probability of detecting an event at a set of stations.

EvLoc and *ARS* use *libmagnitude* in network-magnitude mode to estimate station- and network-magnitude data. As previously discussed, *EvLoc* reads station and event data from an input database, sends them along with control parameters to multiple *libmagnitude* processes, and writes the resulting station- and network-magnitude data to an output database. *ARS* calls *libmagnitude* in much the same way, except that the resulting station- and network-magnitude data are not written to the output database until the analyst explicitly saves the analyzed event [IDC6.5.1]. In other words, multiple sets of station- and network-magnitude data may be generated using different magnitude control parameters, but only the analyst's preferred solution of the set is saved to the output database. *ARS* also calls many more *libmagnitude* processing units than *EvLoc*. Most of these interfaces are low level. Refer to "libmagnitude Processing Units" on page 62 and [IDC6.5.1] for more details on these lower-level interfaces.

Interface with External Users

libmagnitude does not have an interface for external users.

Interface with Operators

libmagnitude writes error messages to standard error. These messages may provide clues to the operator if processing fails. *libmagnitude* also writes various quantities of station and network magnitude results to standard output depending on the level of the verbosity parameter. These results may help to tune the configuration of the calling application. Both error and informational messages are usually redirected to log files during normal IDC operational processing.

Chapter 3: Detailed Design

This chapter describes the detailed design of *EvLoc* and *libmagnitude* and includes the following topics:

- EvLoc Data flow Model
- libmagnitude Data Flow Model
- EvLoc Processing Units
- libmagnitude Processing Units
- Primary libmagnitude Functional Areas
- Data Description

Chapter 3: Detailed Design

EVLOC DATA FLOW MODEL

EvLoc is an application that reads control data from one or more input parameter files, acquires data from an input database, exchanges these data with *libmagnitude* and *libloc* processing units, and records the returned event data in an output database. This brief summary shows that *EvLoc* is primarily a bridge between the database and the central global library processing units.

This chapter tabulates the components of each memory store used by *EvLoc* as each store is introduced. The first column in each table (Tables 5 through 11) lists the name of the component. The names of nested components are indented and marked (">"), and their parent components are listed in the Description column. The second column indicates the data storage type. Table 4 describes each of the possible types of data stored in the memory store components. The third column describes the contents of the components. The fourth column (DD) indicates whether the component is described further in "Data Description" on page 120. The scope of each memory store is also noted as it is introduced. Each memory store is classified either as "internal" or "external" in scope. A memory store is "internal" if data in the store are stored, updated, or read solely within the application or library. A memory store is "external" if data in the store are stored, updated, or read by other applications or software libraries.

TABLE 4: DESCRIPTION OF DATA STORAGE TYPES

Type	Description
variables	collection of numeric/character variables
simple structure	C structure containing only numeric variables, character variables, arrays of numeric variables, or arrays of character variables
complex structure	C structure containing at least one nested structure
nested structure	C structure nested as a member of a complex structure (a nested structure may be simple or complex)
array	array of numeric variables, character variables, structures, or pointers
linked list	linked list of C structures (a linked list may be simple or complex and may be nested)

Figure 7 shows how the data flows into, through, and out of *EvLoc*. In this figure and subsequent data flow diagrams, data flow lines are labeled with the database tables and memory store components that are used as input to and output from the processes. If an entire memory store component is not used, then the data flow line is labeled with the appropriate nested components. Some memory store components shown in Figure 7 are not discussed in this section, but are included in the tables and discussed in "EvLoc Processing Units" on page 49.

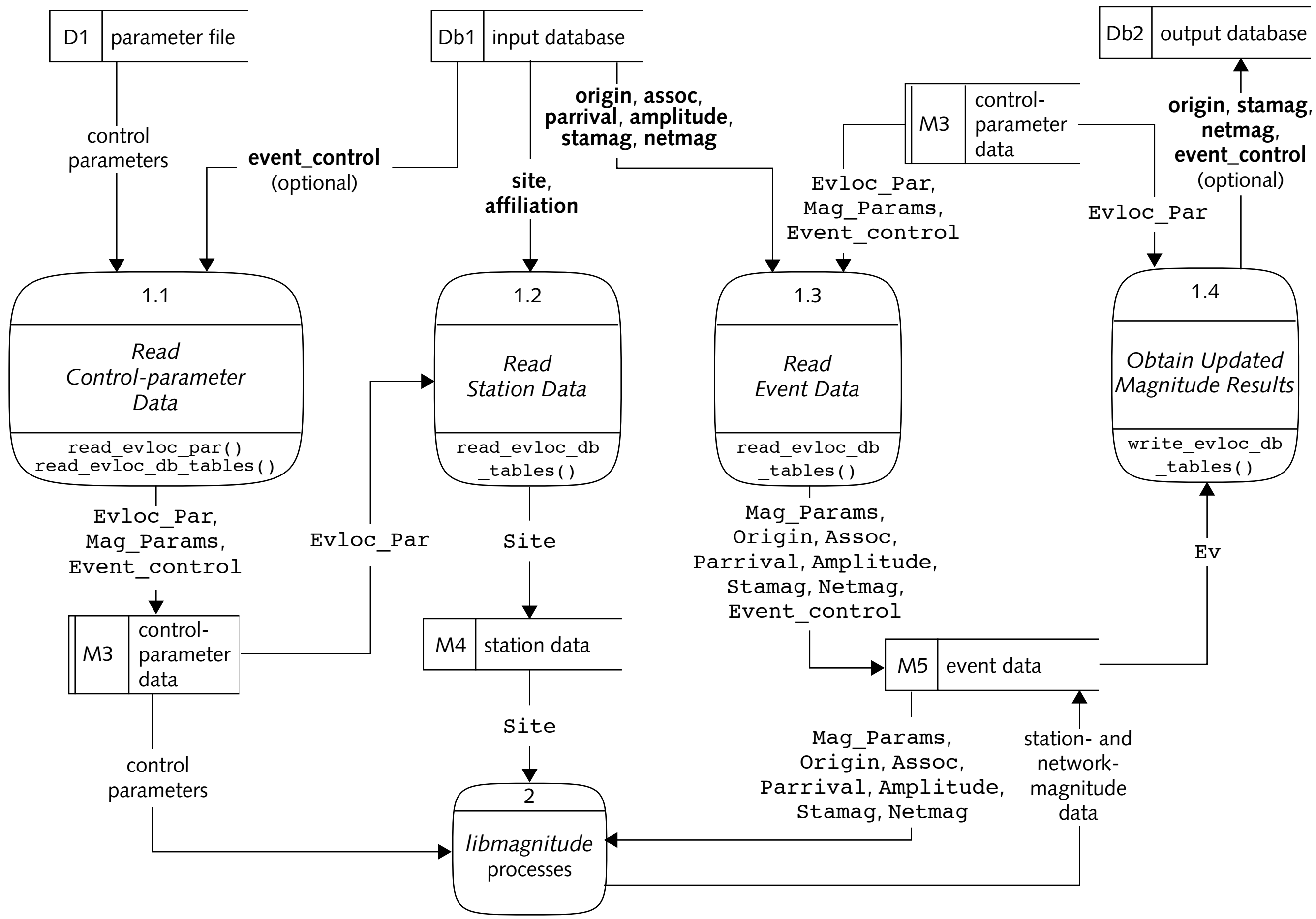


FIGURE 7. EVLOC DATA FLOW MODEL

The first process (1.1 in Figure 7) reads general control-parameter data from an input parameter file (D1) and optionally reads magnitude control parameter data from an **event_control** table in an input database account (Db1). The magnitude control parameters are specified on an event-by-event basis in the **event_control** table. The settings of the optional magnitude control parameters supersede the settings of any corresponding default or general control parameters. The complete set of control parameters control what input data are used and how subsequent processing is performed. Process 1.1 stores all control parameters in an internal control-parameter-data memory store (M3). Table 5 describes the components of this memory store. `read_evloc_par()` reads and stores the data from the parameter file, while `read_evloc_db_tables()` reads and stores the optional data from the **event_control** table.

TABLE 5: CONTROL-PARAMETER-DATA MEMORY STORE (M3)

Component	Data Storage Type	Description of Contents	DD
`EvLoc_Par`	simple structure	*EvLoc* control parameters	yes
`Mag_Params`	simple structure	magnitude control parameters used during processing of magnitude data	yes
`Event_control`	simple structure	event location and magnitude control parameters (optional)	no

The second process (1.2 in Figure 7) reads station data from Db1. Control parameters from the `EvLoc_Par` structure in M3 identify the input **site** and **affiliation** database tables (Table 1 on page 13) and constrain the retrieved station data. The station data are stored in an external station-data memory store (M4). Table 6 describes the components of this memory store. `read_evloc_db_tables()` performs all of this processing.

TABLE 6: STATION-DATA MEMORY STORE (M4)

Component	Data Storage Type	Description of Contents	DD
`Site`	simple structure	station location data	no

The third process (1.3 in Figure 7) reads event data from Db1. Control parameters from the `EvLoc_Par` structure in M3 identify the input **origin**, **assoc**, **parrival**, **amplitude**, **stamag**, and **netmag** database tables (Table 1 on page 13) and constrain the retrieved event data. The event data are stored in an external event-data memory store (M5). Table 7 describes the components of this memory store. The `Mag_Params` and `Event_control` structures are also copied from M3 into M5. `read_evloc_db_tables()` performs all of this processing.

TABLE 7: EVENT-DATA MEMORY STORE (M5)

Component	Data Storage Type	Description of Contents	DD
`Ev`	complex linked list	variables and structures of event data used to estimate and store locations/magnitudes for one or more events; each event is represented by a single complex structure in a linked list	yes
`> Origin`	simple nested structure	origin for a particular event; nested within the `Ev` linked list	no
`> Assoc`	simple nested structure	connects arrivals to a particular origin; nested within the `Ev` linked list	no
`> Event_control`	simple nested structure	event location and magnitude control parameters (optional); nested within the `Ev` linked list	no
`> Mag_ptr`	complex nested structure	variables and structures of the event (primarily magnitude) data; nested within the `Ev` linked list	yes
`>> Magnitude`	complex nested structure	station amplitude and magnitude data, network-magnitude data, and magnitude specification data; nearly all of the structure members are populated in *libmagnitude*, nested within the `Mag_ptr` structure	yes
`> Mag_Params`	simple nested structure	general magnitude control parameters used during processing of magnitude data; nested within the `Ev` linked list	yes

TABLE 7: EVENT-DATA MEMORY STORE (M5) (CONTINUED)

Component	Data Storage Type	Description of Contents	DD
`Parrival`	simple structure	predicted arrivals and associations for origin-based amplitude measurements	no
`Amplitude`	simple structure	amplitude measurements	no
`Stamag`	simple structure	station-magnitude estimates	no
`Netmag`	simple structure	network-magnitude estimates	no

The *libmagnitude* processing units read data subsets from M3, M4, and M5 and estimate station/network magnitudes. *libmagnitude* stores the resulting station- and network-magnitude data in `Magnitude` objects (see Table 54 on page 133). A `Magnitude` object is a nested structure within the `Mag_ptr` structure, which itself is a nested structure within the `Ev` linked list in M5. As a result, data in `Magnitude` objects are accessible by *EvLoc*. Refer to "libmagnitude Data Flow Model" for additional details regarding the data flow into, through, and out of *libmagnitude*.

The fourth process (1.4 in Figure 7) obtains updated station- and network-magnitude data from the `Ev` linked list in M5 and writes it to an output database (Db2). Control parameters from the `EvLoc_Par` structure in M3 identify the output database tables (Table 1 on page 13). The station- and network-magnitude data are copied from the `Stamag` and `Netmag` structures (stored within the `Magnitude` objects) and the `Origin` structures (stored within the `Ev` linked list) into analogous database tables in the output database. `write_evloc_db_tables()` performs all of this processing.

LIBMAGNITUDE DATA FLOW MODEL

The *libmagnitude* software library handles all magnitude-related computations and processing. As mentioned in "Conceptual Design" on page 10, the two primary *libmagnitude* modes of operation are (1) estimating station-magnitude data and (2) estimating network-magnitude data. This section describes how the data flows into, through, and out of *libmagnitude* in each mode of operation. In addition, the components of each *libmagnitude* memory store will be tabulated as each store is introduced. The tables are structured analogously to those in "EvLoc Data flow Model" on page 30. Refer to that section for descriptions of the columns associated with each table.

As with the *EvLoc* data flow diagram (Figure 7 on page 32), if an entire memory store component is not used as input to or output from a process in the *libmagnitude* data flow diagrams, then the corresponding data flow line is labeled only with the appropriate nested components. Some memory store components shown in the *libmagnitude* data flow diagrams are not discussed in this section, but are included in the tables and discussed in "libmagnitude Processing Units" on page 62.

Station-magnitude Mode

When an application operates in station-magnitude mode, *libmagnitude* estimates a single station magnitude, uncertainty, and additional data given a single amplitude and magtype. *libmagnitude* records these data in an external memory store, which is accessible by the calling application upon completion of *libmagnitude* processing.

Figure 8 shows how the data flows into, through, and out of *libmagnitude* in station-magnitude mode. The calling application must first store station data, event data, and control parameters in an external station, event, and control-parameter data memory store (M6) before any *libmagnitude* processing can occur. The station data reside in `Site` database table structures. The event data and control parameters are stored in a number of constructs that are unique to each application. Typically these constructs are database table structures, such as `Origin`, `Assoc`, `Parrival`, and `Amplitude`, along with one or more simple structures that store the control parameters. Because the components of M6 vary by application, they are not tabulated here.

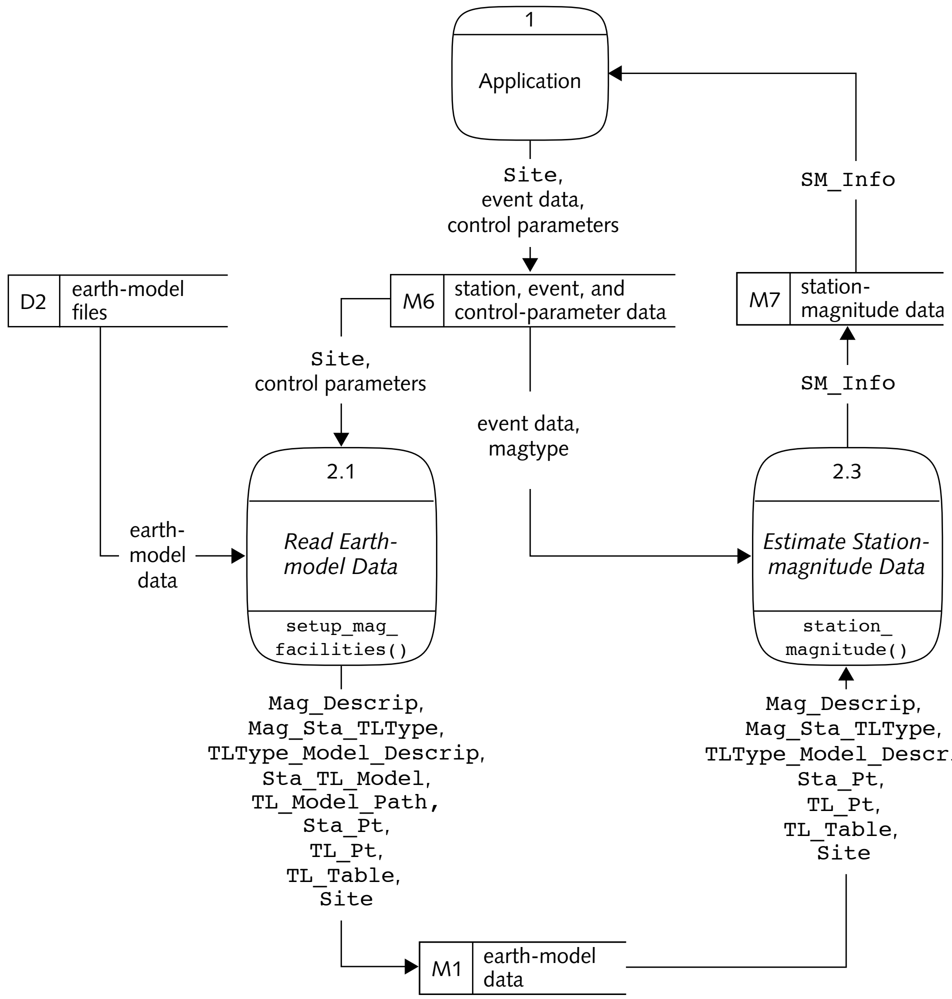


FIGURE 8. LIBMAGNITUDE DATA FLOW MODEL IN STATION-MAGNITUDE MODE

The first process (2.1 in Figure 8) reads earth-model data from a set of earth-model files (D2) and stores the data in a memory store (M1). Station data and a subset of control parameters from M6 are used to identify the proper earth-model files and constrain the retrieved data. The files are composed of an MDF, a TLSF, and one or more TLMs. Earth-model data read from D2 are stored in a primarily internal earth-model-data memory store (M1). Table 8 describes the components of this data store. Additional details of the M1 components are provided in "libmagnitude Processing Units" on page 62 under the processing units that compose the *Read Earth-model Data* process. The `Mag_Descrip` structure is an external component of M1 because applications may change the values of its members (see "read_ mdf()" on page 77 for more details). This design feature is not indicated in Figure 8.

TABLE 8: EARTH-MODEL-DATA MEMORY STORE (M1)

Component	Data Storage Type	Description of Contents	DD
`Mag_Descrip`	array of simple structures	magnitude description data	yes
`Mag_Sta_TLType`	array of simple structures	bulk-station-correction data	yes
`TLType_Model_Descrip`	complex structure	default TLM description data	yes
`> List_of_Phz`	simple nested linked list	linked list of phase names nested within structure `TLType_Model_Descrip`	no
`Sta_TL_Model`	simple structure	station-specific TLM description data	yes
`TL_Model_Path`	simple structure	TLM pathway data	yes
`Sta_Pt`	complex structure	pointer to a simple linked list of `TL_Pt` structures	no
`TL_Pt`	simple linked list or simple nested linked list	linked list of TLtypes or linked list of subset of station-specific TLM description data; nested within `Sta_Pt`	no

TABLE 8: EARTH-MODEL-DATA MEMORY STORE (M1) (CONTINUED)

Component	Data Storage Type	Description of Contents	DD
`TL_Table`	complex structure	transmission-loss and transmission-loss modeling-error data	yes
> `TL_Mdl_Cor`	simple nested structure	transmission-loss modeling-error data; nested within the `TL_Table` structure	yes
> `TL_TS_Cor`[1]	simple nested structure	test-site correction data; nested within the `TL_Table` structure	no
`Site`	simple structure	station location data	no

1. This structure is not applicable to the IDC.

`setup_mag_facilities()` is the *libmagnitude* interface associated with the *Read Earth-model Data* process. `setup_mag_facilities()` is typically only called one time by an application. After the earth-model data are stored in memory, several *libmagnitude* processing units are available for applications to access these data. This function is required in both station- and network-magnitude modes.

The *Estimate Station-magnitude Data* process (2.3 in Figure 8) calculates a single station magnitude, uncertainty, and additional station-magnitude data for a given amplitude and magtype. Event data and a single magtype from the station, event, and control-parameter data memory store (M6) are used to estimate an initial station magnitude and to identify the earth-model data that should be retrieved from several components of the earth-model-data memory store (M1; Table 8). This initial station magnitude is revised to produce a final station magnitude by applying magnitude corrections retrieved from the earth-model data. The station-magnitude uncertainty is calculated as the root-mean-square of several error estimates that are also retrieved from the earth-model data. The station magnitude, uncertainty, and additional station-magnitude data (which include the applied magnitude corrections and error estimates) are stored in an external station-magnitude-data memory store (M7 in Figure 8). Table 9 describes the components of this memory store.

TABLE 9: STATION-MAGNITUDE-DATA MEMORY STORE (M7)

Component	Data Storage Type	Description of Contents	DD
`SM_Info`	simple structure	station-magnitude data	yes

`station_magnitude()` is the *libmagnitude* interface associated with the *Estimate Station-magnitude Data* process. `station_magnitude()` should be called multiple times to process multiple amplitudes and magtypes associated with a single event.

Figure 9 shows how the data flows into, through, and out of the *Read Earth-model Data* process (2.1 in Figure 8), regardless of the mode of operation. The external interface to this process is `setup_mag_facilities()`. While not explicitly shown in Figure 9, `setup_mag_facilities()` calls other *libmagnitude* processing units that in turn call `read_mdf()`, `read_tlsf()`, and `read_tl_table()` (see Table 18 on page 63). These three functions read, parse, and store data from the MDF, TLSF, and one or more TLMs, respectively. These functions are assigned unique subprocess identifiers of the form "2.1.*n*" in Figure 9. The disk stores D2.a, D2.b, and D2.c comprise the earth-model data disk store (D2 in Figure 8).

The first subprocess (2.1.1 in Figure 9) reads and parses magnitude description (magnitude control) and bulk station-correction data from the MDF (D2.a) and stores these data in the earth-model-data memory store (M1). An MDF pathname and a list of magtypes from the station, event, and control-parameter data memory store (M6) identify which MDF is to be read and what data within the MDF are stored in M1. `read_mdf()` performs this processing.

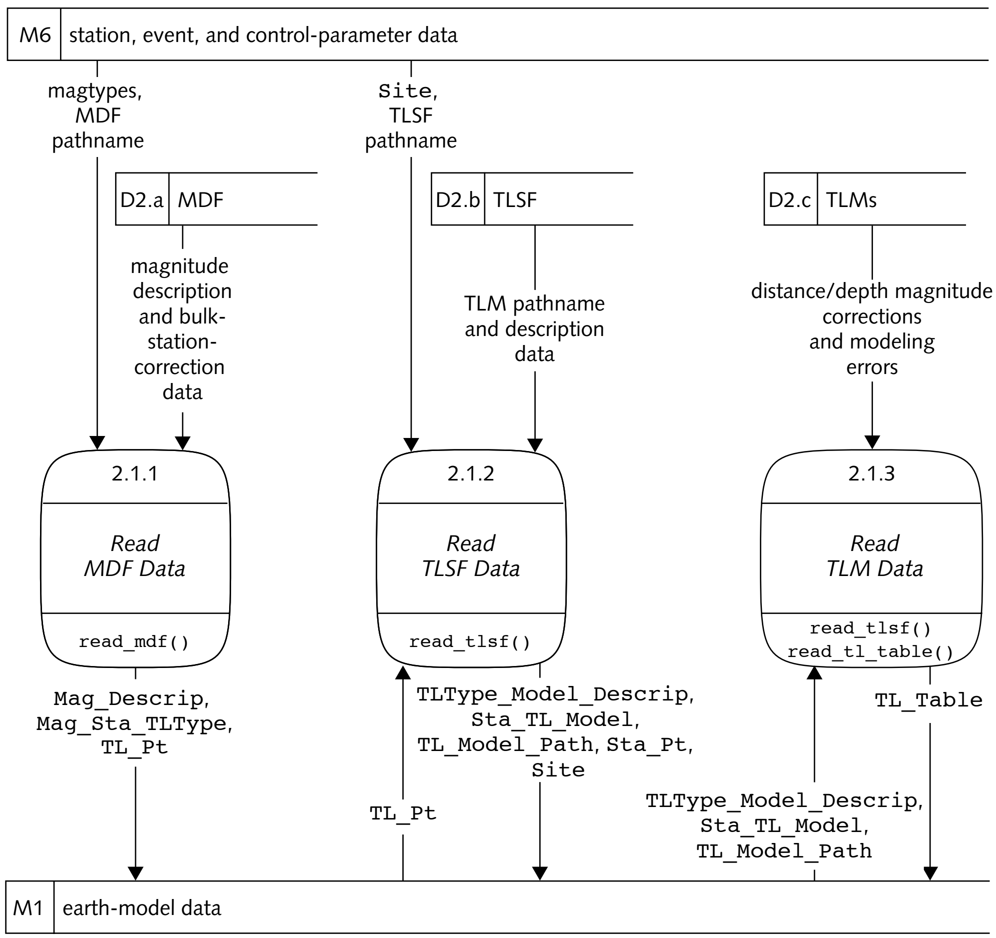


FIGURE 9. DETAILED DATA FLOW OF READ EARTH-MODEL DATA

The second subprocess (2.1.2 in Figure 9) reads and parses TLM pathway data, default TLM description data, and station-specific TLM description data from the TLSF (D2.b) and stores these data in M1. A TLSF pathname from M6 identifies which TLSF is to be read. Station data from M6 and a list of TLtypes from M1 identify what data within the TLSF are stored in M1. `read_tlsf()` performs this processing.

The third subprocess (2.1.3 in Figure 9) reads and parses distance/depth-dependent transmission-loss (that is, magnitude correction) data and modeling errors from one or more TLMs (D2.c) and stores these data in M1. Components (root names and suffixes) necessary to construct the full path names of the TLMs are read from the TLM pathway data and TLM description data stored in M1. `read_tlsf()` and `read_tl_table()` perform this processing.

Network-magnitude Mode

When an application operates in network-magnitude mode, *libmagnitude* is used to estimate station and network magnitudes, uncertainties, and additional magnitude data. *libmagnitude* stores these data in two external memory stores that may be accessed by the calling application upon completion of *libmagnitude* processing. The station-magnitude mode processes are required in network-magnitude mode.

Figure 10 shows how the data flows into, through, and out of *libmagnitude* in the network-magnitude mode. The calling application must first store station data, event data, and control parameters in a station, event, and control-parameter data memory store (M6). Table 10 describes the components of this memory store. The general control parameters are stored in constructs unique to each application. In the case where *EvLoc* is the calling application, the station-data memory store (M4 in Figure 7 on page 32; Table 6 on page 33) and portions of the control-parameter-data memory store (M3 in Figure 7; Table 5 on page 33) and event-data memory store (M5 in Figure 7; Table 7 on page 34) combine to form M6.

The *Read Earth-model Data* process (2.1 in Figure 10) reads earth-model and control data from a set of earth-model files (D2) and stores these data in an internal earth-model-data memory store (M1; Table 8 on page 38). The description of the data flow into, through, and out of this process in network-magnitude mode is identical to that in station-magnitude mode (Figure 8 on page 37). `setup_mag_facilities()` is the *libmagnitude* interface associated with the *Read Earth-model Data* process. It must be called prior to any other *libmagnitude* process.

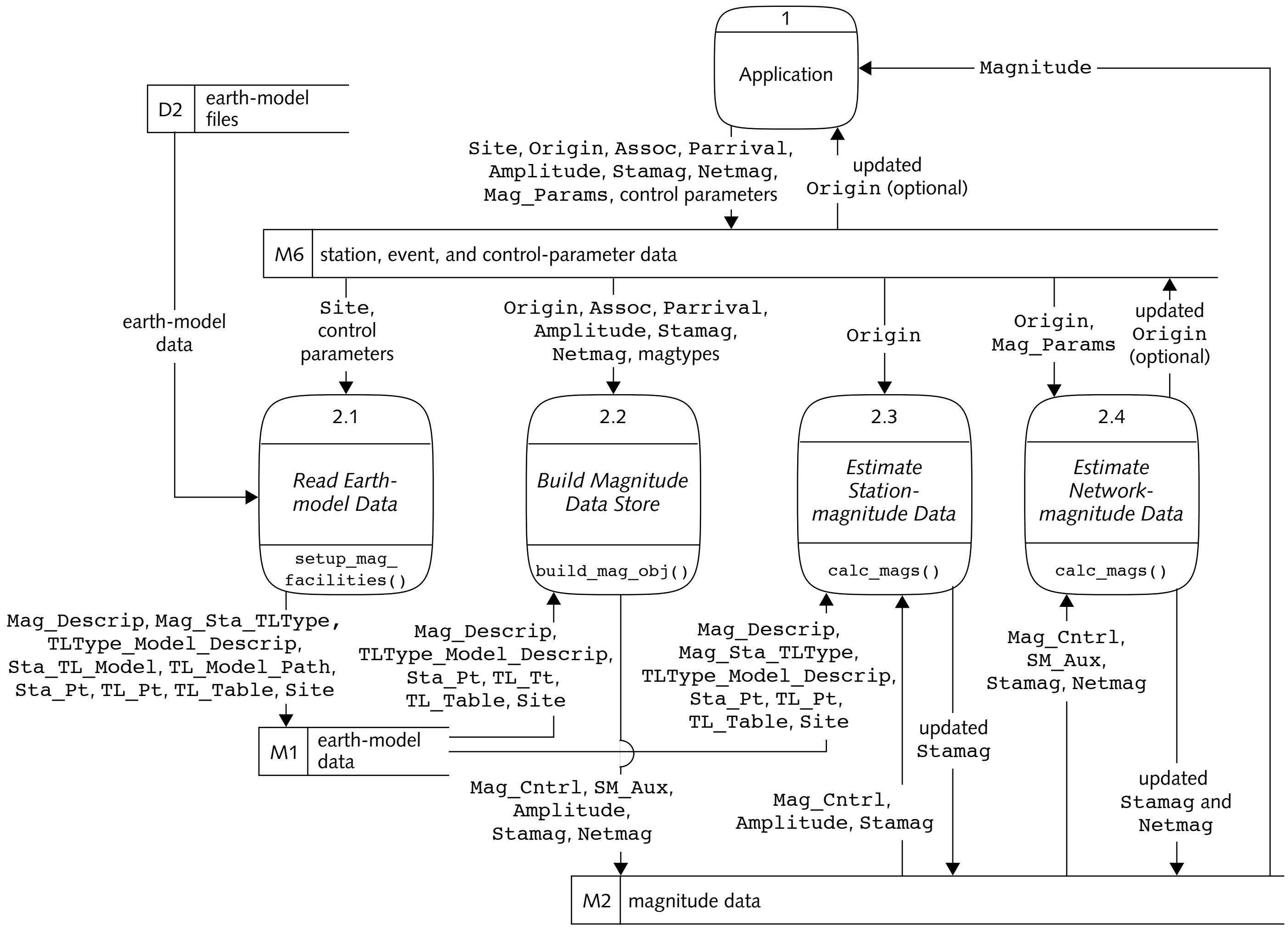


FIGURE 10. LIBMAGNITUDE DATA FLOW IN NETWORK-MAGNITUDE MODE

The *Build Magnitude Data Store* process (2.2 in Figure 10) stores event and magnitude specification data on an event-by-event basis in memory store M2. A list of magtypes from the control-parameter variables stored in M6 are used to constrain which event data from M6 are stored in M2. Table 10 describes these components. The constrained event data, which are primarily elements of the `Amplitude`, `Stamag`, and `Netmag` database table structures, are stored within an array of `Magnitude` objects (Table 54 on page 133) in an external magnitude-data memory store (M2). Table 11 describes the components of this memory store. The list of magtypes also identifies which magnitude description data in the `Mag_Descrip` structures (M1; Table 8 on page 38) are copied and stored as magnitude specification data in the `Mag_Cntrl` structures within the array of `Magnitude` objects. The consolidation of data from M6 and M1 into M2 permits convenient access to the data during the station- and network-magnitude estimation processes.

TABLE 10: STATION, EVENT, AND CONTROL-PARAMETER DATA MEMORY STORE (M6)

Component	Data Storage Type	Description of Contents	DD
`Site`	simple structure	station location data	no
`Origin`	simple structure	origin information for particular event	no
`Assoc`	simple structure	arrival association information	no
`Parrival`	simple structure	predicted arrivals and associations for origin-based amplitude measurements	no
`Amplitude`	simple structure	amplitude measurements	no
`Stamag`	simple structure	station-magnitude estimates	no
`Netmag`	simple structure	network-magnitude estimates	no
`Mag_Params`	simple structure	magnitude control parameters used during processing of magnitude data	yes
control parameters	variables	general control parameters	no

TABLE 11: MAGNITUDE-DATA MEMORY STORE (M2)

Component	Data Storage Type	Description of Contents	DD
`Magnitude`	complex structure	station amplitude and magnitude data, network-magnitude data, and magnitude specification data	yes
> `Mag_Cntrl`	simple nested structure	magnitude specification data; nested within `Magnitude` object	no[1]
> `SM_Aux`	simple nested structure	auxiliary station-magnitude data; nested within `Magnitude` object	yes
> `Amplitude`	simple nested structure	amplitude measurements; nested within `Magnitude` object	no
> `Stamag`	simple nested structure	station-magnitude estimates; nested within `Magnitude` object	no
> `Netmag`	simple nested structure	network-magnitude estimates; nested within `Magnitude` object	no
`SM_Sub`[2]	simple structure	station-magnitude data subset	yes

1. This structure contains a subset of the data stored in the `Mag_Descrip` structure.
2. This structure is used to pass station-magnitude data between processing units in the *Estimate Network-magnitude Data* process

`build_mag_obj()` is the *libmagnitude* interface associated with the *Build Magnitude Data Store* process. `build_mag_obj()` builds a `Magnitude` object for each event with assistance from other lower-level *libmagnitude* processing units. `build_mag_obj()` should be called once for each distinct event processed. The data in a `Magnitude` object, particularly the magnitude specification data and auxiliary station-magnitude data, may be optionally revised for each event by the application after completion of this process, but before execution of the *Estimate Station-magnitude Data* process and the *Estimate Network-magnitude Data* process (2.3 and 2.4, respectively). This is a design feature for `Magnitude` objects. This feature is not indicated in Figure 10.

The *Estimate Station-magnitude Data* process (2.3 in Figure 10) estimates station magnitudes, uncertainties, and additional station-magnitude data for multiple amplitudes and magtypes associated with a single event. The description of the data flow into, through, and out of this process is similar to that in station-magnitude mode (Figure 8 on page 37). The inputs to this process are the same as in station-magnitude mode, but the data are retrieved from different memory stores. The source for most of the event data is the magnitude-data memory store (M2; Table 11), but the event location comes from the `Origin` structure in the station, event, and control-parameter data memory store (M6; Table 10). The data source for the list of magtypes is the `Mag_Cntrl` structure in M2. The source for the earth-model data is the earth-model-data memory store (M1; Table 8 on page 38), as it was in station-magnitude mode.

The station-magnitude data are estimated the same way as described for the *Estimate Station-magnitude Data* process in station-magnitude mode (2.3 in Figure 8 on page 37). However, the station-magnitude data are not structured or stored in quite the same way. The station-magnitude-data memory store (M7 in Figure 8; Table 9 on page 40) is still populated within the *Estimate Station-magnitude Data* process (2.3 in Figure 10), but it is not accessed by other *libmagnitude* processes or external applications. As a result, the station-magnitude-data memory store is an internal memory store and is not shown in Figure 10. The station magnitudes and uncertainties estimated in the *Estimate Station-magnitude Data* process in network-magnitude mode are stored in the `Stamag` database table structures within M2.

`calc_mags()` is the *libmagnitude* interface associated with the *Estimate Station-magnitude Data* process. `calc_mags()` should be called once for each distinct event processed. `calc_mags()` in turn calls `station_magnitude()` for each valid amplitude datum for a given magtype. Multiple network magnitudes may be determined for a single event.

The *Estimate Network-magnitude Data* process (2.4 in Figure 10) estimates network-magnitude data for multiple magtypes associated with a single event and stores the results in two memory stores. Governed by the control parameters retrieved from M6 (Table 10) and the magnitude specification data retrieved from M2 (Table 11), the processing units within this process estimate network magnitudes, uncertainties, and additional magnitude data using station magnitudes and uncertainties retrieved from M2. The network-magnitude uncertainties are standard deviations of means of the input station magnitudes. The resulting network magnitudes, uncertainties, and ancillary network-magnitude data are stored in the `Netmag` database table structures within M2. The resulting network magnitudes are also optionally written to `Origin` database table structures retrieved from M6. The station-magnitude residuals are stored in the `Stamag` database table structures within M2.

`calc_mags()` is also the *libmagnitude* interface associated with the *Estimate Network-magnitude Data* process. `calc_mags()` estimates network-magnitude data with assistance from other lower-level *libmagnitude* processing units. `calc_mags()` should be called once for each event.

Figure 11 shows how the data flows into, through, and out of the *Estimate Network-magnitude Data* process (2.4 in Figure 10). The external interface to this process is `calc_mags()`. `calc_mags()` in turn calls `network_mag()`, `only_bound_amps()`, `mag_max_lik()`, and `mag_boot_strap()` to estimate network-magnitude data using various magnitude algorithms. These functions are assigned unique subprocess identifiers of the form "2.4.*n*" in Figure 11. The inputs to and outputs from each process are nearly identical to the inputs and outputs described previously for the *Estimate Network-magnitude Data* process (2.4). The only difference is the type of network-magnitude data estimated (that is, what algorithm was used) by each process.

The first subprocess (2.4.1 in Figure 11) estimates network-average magnitudes and uncertainties for one or more magtypes associated with a single event. The network-average magnitude data and corresponding station-magnitude residuals are stored in the magnitude-data memory store (M2 in Figure 11). The network-average magnitudes are optionally stored in the station, event, and control-parameter data memory store (M6 in Figure 11). `network_mag()` and `calc_mags()` perform this processing.

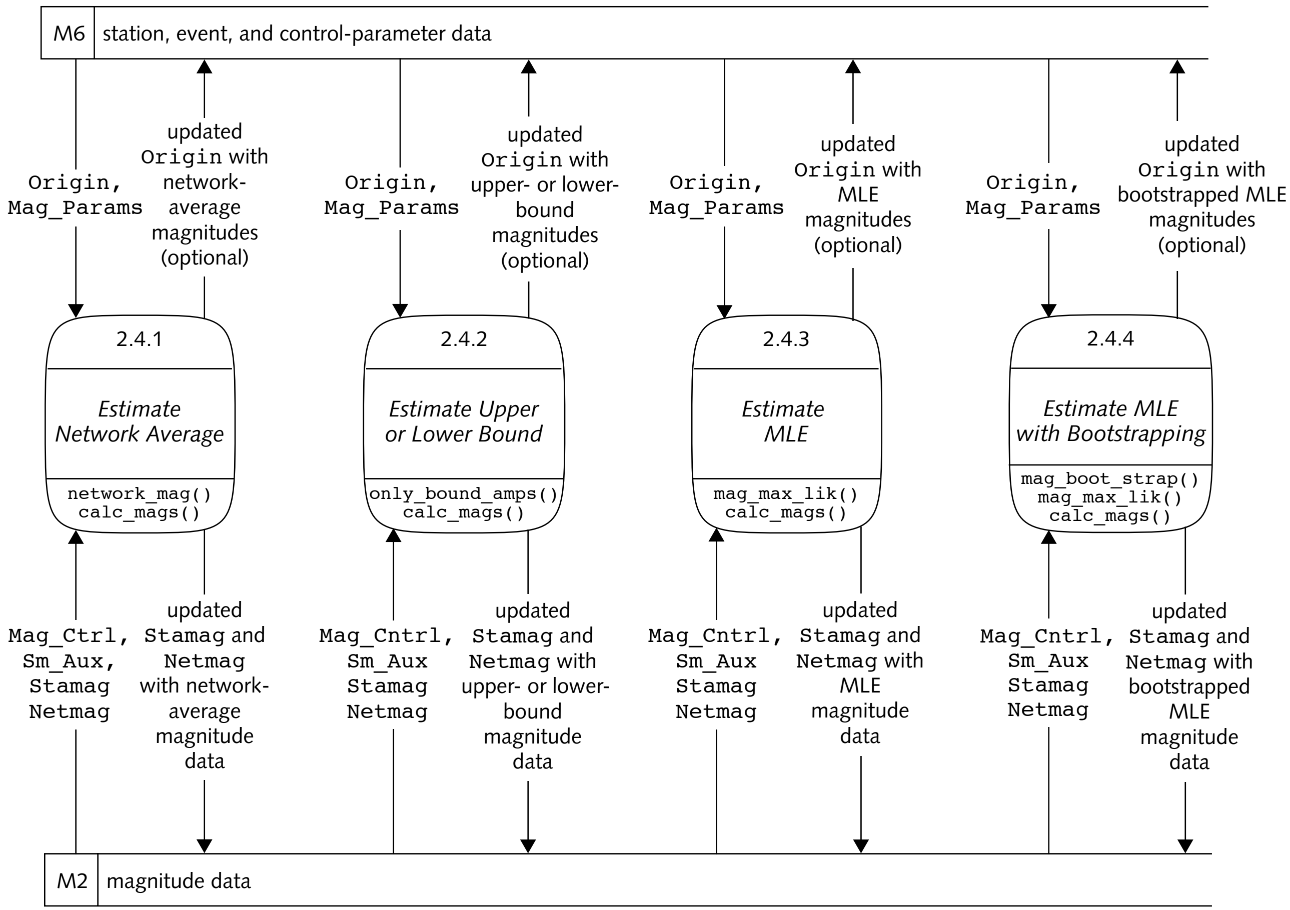


FIGURE 11. DETAILED DATA FLOW OF ESTIMATE NETWORK-MAGNITUDE DATA

The second subprocess (2.4.2 in Figure 11) estimates upper- or lower-bound magnitudes and uncertainties for one or more magtypes associated with a single event. The upper- or lower-bound magnitude data and corresponding station-magnitude residuals are stored in M2. The upper- or lower-bound magnitudes are optionally stored in M6. `only_bound_amps()` and `calc_mags()` perform this processing.

The *Estimate MLE* process (2.4.3 in Figure 11) estimates MLE magnitudes and uncertainties for one or more magtypes associated with a single event. The MLE-magnitude data and corresponding station-magnitude residuals are stored in M2. The MLE magnitudes are optionally stored in M6. `mag_max_lik()` and `calc_mags()` perform this processing.

The fourth subprocess (2.4.4 in Figure 11) estimates MLE magnitudes and uncertainties via bootstrap resampling [McL88] for one or more magtypes associated with a single event. The bootstrapped MLE-magnitude data and corresponding station-magnitude residuals are stored in M2. The bootstrapped MLE magnitudes are optionally stored in M6. `mag_boot_strap()`, `mag_max_lik()`, and `calc_mags()` perform this processing.

EVLOC PROCESSING UNITS

The *EvLoc* portion of the Event Magnitude software includes nine processing units, which are listed in Table 12. This table lists the hierarchy of the processes with respect to one another and to processing units from several global software libraries. The table structure follows the control flow: the highest-level processing units are listed first, followed by the lower-level processing units in the order they are called.

The four most important processing units related to magnitude processing are listed below. The following paragraphs describe the design of these units, including any constraints or unusual features in the design. The logic of the software and any applicable procedural commands are also provided. The remaining five processing units, listed in Table 12, perform simple or low-level specialized tasks or sets of tasks and are not described further.

- `main()`

- `read_evloc_par()`
- `read_evloc_db_tables()`
- `write_evloc_db_tables()`

The description of each processing unit includes a table that describes its input and output variables. Each table lists the name and data type and describes each input or output variable. In addition, the Use column assigns each variable a classification that indicates how the variable is used. The value is "A" if the variable is input or output through the processing unit's argument list and "R" if the variable is the return value from the processing unit.

TABLE 12: HIERARCHY OF EVLOC PROCESSING UNITS

Processing Unit	Description	Called from	Calls to
`main()`	estimates event locations and magnitudes through calls to other *EvLoc* and global library functions	command line	`read_evloc_par()`, `read_evloc_db_tables()`, `write_evloc_db_tables()`, `make_predicts()`; *libmagnitude*, *libloc*, *libpar*, and *libstdtime* functions
`read_evloc_par()`	reads control-parameter data	`main()`	`evloc_init_par()`; *libmagnitude*, *libloc*, and *libpar* functions
`read_evloc_db_tables()`	reads station, event, and control data from input database tables	`main()`	`setup_ev_cntrl_table()`, `reset_loc_controls()`, `reset_mag_controls()`; *libmagnitude*, *libloc*, and *libgdi* functions
`make_predicts()`	computes theoretical travel-time, azimuth, and slowness data for detections associated to event	`main()`	`write_evloc_db_tables()`; *libloc* and *libgeog* functions

TABLE 12: HIERARCHY OF EVLOC PROCESSING UNITS (CONTINUED)

Processing Unit	Description	Called from	Calls to
`write_evloc_db_tables()`	writes updated event and control data to output database tables	`main()`, `make_predicts()`	*libgdi* and *libstdtime* functions
`evloc_init_par()`	initializes control-parameter structure (see Table 5 on page 33) to the default values	`read_evloc_par()`	none
`setup_ev_cntrl_table()`	initializes event-location and magnitude control structures (see Table 7 on page 34) to the default values	`read_evloc_db_tables()`	none
`reset_loc_controls()`	changes locator control-parameter settings	`read_evloc_db_tables()`	none
`reset_mag_controls()`	changes magnitude control parameter (see Table 7 on page 34) settings	`read_evloc_db_tables()`	*libmagnitude* functions

main()

`main()` is the primary *EvLoc* processing unit. It calls other *EvLoc* processing units and global library functions to estimate magnitude data.

Input/Processing/Output

`main()` is the highest-level *EvLoc* processing unit. It initiates four *EvLoc* processes (Figure 7 on page 32) to estimate station- and network-magnitude data.

Table 13 describes the input variables to `main()`. These variables are command line arguments.

TABLE 13: INPUT VARIABLES TO MAIN()

Type	Variable Name	Use	Description
`int`	*argc*	A	number of fields on command line
`char **`	*argv*	A	array of character strings entered on command line and separated by spaces

A typical command line for executing *EvLoc* is:

```
EvLoc par=EvLoc.par
```

where `EvLoc.par` is the name of an input parameter file containing control-parameter arguments.

`main()` calls `read_evloc_par()` (see Table 12 for hierarchy) to read control-parameter data from an input parameter file. It also calls `read_evloc_db_tables()` to read station and event data and optional magnitude control parameters from input database tables. `main()` calls *libmagnitude* functions to estimate magnitude data. It obtains station- and network-magnitude results from *libmagnitude* functions and updates members of magnitude-related structures. Finally, `main()` calls `write_evloc_db_tables()` to write the resulting magnitude data to output database tables.

`main()` terminates with a status code upon completion of processing.

Interfaces

`main()` calls the lower-level *EvLoc* processing units listed in Table 12.

`main()` also calls processing units from several global libraries (Table 12). It calls the *libmagnitude* processing unit `calc_mags()` to estimate station- and network-magnitude data. `main()` also calls the *libloc* processing unit `locate_event()` to determine event hypocenters. `main()` calls the *libpar* processing unit `setpar()` to parse environment and command line arguments, and `endpar()` to clean up memory associated with reading the input parameter file. `main()` also calls the *libstdtime* processing unit `stdtime_get_epoch()` to retrieve the current system time in epoch seconds. Refer to the *libpar* and *libstdtime* man pages for descriptions of the interfaces.

Error States

`main()` interprets the status codes or return values returned from the *EvLoc* processing units `read_evloc_par()` and `read_evloc_db_tables()` (see Table 12 for hierarchy). It also interprets the status codes or return values from the global library functions: `calc_mags()`, `setpar()`, and `stdtime_get_epoch()`. OK status codes or acceptable return values indicate to `main()` that processing was successful. Error status codes or out-of-bounds return values indicate to `main()` that it should terminate all further *EvLoc* processing and write an error message to stderr.

`main()` writes the following error message to stderr and terminates further *EvLoc* processing if the command line does not contain the name of an input parameter file or a list of control parameters:

```
Usage: EvLoc par=par_filename
```

`main()` exits with an OK status code if `read_evloc_db_tables()` does not retrieve any event data from the input database account.

read_evloc_par()

`read_evloc_par()` reads control-parameter data from an input parameter file and stores the data in a memory store.

Input/Processing/Output

`read_evloc_par()` is called by `main()` as part of the *Read Control-parameter Data* process (Figure 7 on page 32); `read_evloc_par()` does not require input variables.

`read_evloc_par()` reads general control parameters from an input parameter file (D1 in Figure 7) and stores them in external memory. `read_evloc_par()` parses the parameter file and stores the control parameters in several structures that include the `EvLoc_Par` (Table 50 on page 125) and `Mag_Params` structures (Table 51 on page 129). Only the `EvLoc_Par` and `Mag_Params` structures are discussed in this document because they store general and magnitude-specific control parameters that control magnitude determinations. Refer to [IDC7.1.5] for descriptions of other structures populated by `read_evloc_par()` that contain event location control parameters.

`read_evloc_par()` returns the output variables listed in Table 14 to `main()`. The `EvLoc_Par` and `Mag_Params` structures are stored as components of the control-parameter-data memory store (M3 in Figure 7 on page 32; Table 5 on page 33).

TABLE 14: OUTPUT VARIABLES FROM READ_EVLOC_PAR()

Type	Variable Name	Use	Description
`int`	*icode*	R	status code
`EvLoc_Par *`	*evloc_par*	A	pointer to `EvLoc_Par` structure
`Mag_Params *`	*mag_params*	A	pointer to `Mag_Params` structure

Interfaces

Only `main()` calls `read_evloc_par()` within *EvLoc* (see Table 12). `read_evloc_par()` requires that the lower-level *EvLoc* initialization function, `evloc_init_par()`, be called.

`read_evloc_par()` also calls processing units from several global libraries (Table 12). It calls the *libmagnitude* processing unit `initialize_mag_params()` to define default settings for the `Mag_Params` structure and the *libloc* processing unit `initialize_loc_params()` to define default settings for the `Locator_params` structure [IDC-7.1.5]. `read_evloc_par()` also calls the *libpar* processing units `getpar()` and `mstspar()` to read control-parameter arguments from the input parameter file. (Refer to the *libpar* man page for descriptions of the interfaces.)

Error States

`read_evloc_par()` writes an error message to stderr and terminates all further *EvLoc* processing if it attempts to read required control parameters that are not present in the input parameter file. Examples of required control parameters are database table names. See the *EvLoc* man page for a complete list of required control parameters.

`read_evloc_par()` writes an error message to stderr, terminates all further processing within itself, and returns an error status code of `–1` to `main()` if more than 200 stations are listed in a substation list.

If `read_evloc_par()` reads, parses, and stores the control parameters from the input parameter file without encountering any error conditions, it returns an OK status code to *main()*.

read_evloc_db_tables()

`read_evloc_db_tables()` reads station and event data and optional magnitude control parameters from an input database and stores the data in two memory stores.

Input/Processing/Output

`read_evloc_db_tables()` is called by *main()* as part of the *Read Control-parameter Data* process, the *Read Station Data* process, and the *Read Event Data* process (Figure 7 on page 32). The input variables to `read_evloc_db_tables()` are shown in Table 15. The data source for these variables is the control-parameter-data memory store (M3 in Figure 7; Table 5 on page 33).

TABLE 15: INPUT VARIABLES TO READ_EVLOC_DB_TABLES()

Type	Variable Name	Use	Description
`EvLoc_Par *`	*evloc_par*	A	pointer to `EvLoc_Par` structure
`Mag_Params *`	*mag_params*	A	pointer to `Mag_Params` structure

`read_evloc_db_tables()` reads station data, event data, and optional magnitude control parameters from an input database account (Db1 in Figure 7) and stores these data in two external memory stores. `read_evloc_db_tables()` uses the Generic Database Interface (GDI) to dynamically link to and establish a connection with the input database, retrieve records from the database, close the connection, and handle any errors that occur during these processes.

`read_evloc_db_tables()` reads station data from a **site** table using two criteria. First, the stations themselves must be affiliated with the station network specified by the `network` member of the `EvLoc_Par` structure (Table 50 on page 125). `read_evloc_db_tables()` retrieves the affiliations themselves from the input **affiliation** table. Second, the stations must be installed and operational during the time period being processed. `read_evloc_db_tables()` stores the station data in an array of `Site` database table structures (see Table 2 on page 14).

`read_evloc_db_tables()` reads event data from the **origin**, **assoc**, **parrival**, **amplitude**, **stamag**, and **netmag** tables. Two **amplitude** tables may be read; one may contain arrival-based amplitude data (that is, amplitude data measured for an arrival), and the other may contain origin-based amplitude data (that is, amplitude data measured in a predicted time window relative to the origin). Both arrival-based and origin-based amplitude data may be contained in the same **amplitude** table.

`read_evloc_db_tables()` stores the event data from the **parrival**, **amplitude**, **stamag**, and **netmag** tables in arrays of `Parrival`, `Amplitude`, `Stamag`, and `Netmag` database table structures, respectively (see Table 2).

read_evloc_db_tables() stores the event data read from the **origin** and **assoc** tables within an Ev linked list (Table 52 on page 130). For each record read from the **origin** table, read_evloc_db_tables() creates an element of the linked list and links it to any previous elements in the list. The **origin** record itself is stored in an Origin database table structure (see Table 2) nested within the newly created element of the Ev linked list. The **assoc** records associated with that origin are stored in an array of Assoc structures also nested within the newly created element.

After a new element is created in the Ev linked list, read_evloc_db_tables() populates the Mag_Params and Mag_ptr component members of the element. read_evloc_db_tables() copies the input Mag_Params structure (Table 51 on page 129) into the Mag_Params component member of the new element. read_evloc_db_tables() calls the *libmagnitude* interface build_mag_obj() to initialize and populate the Magnitude object (Table 54 on page 133) member of the Mag_ptr structure (Table 53 on page 132) within the new element.

read_evloc_db_tables() optionally reads magnitude control parameters from an input **event_control** table. The contents of any **event_control** records override some of the magnitude control parameters set in the Mag_Params member of the element of the Ev linked list corresponding to the event; the Mag_Params structure is updated accordingly. Each **event_control** record itself is also stored in an Event_control database table structure (see Table 2) nested within the element of the Ev linked list corresponding to the event.

read_evloc_db_tables() stores the station and event data and magnitude control parameters in two memory stores. The station data are stored in a station-data memory store (M4 in Figure 7 on page 32; Table 6 on page 33). The event data and magnitude control parameters are stored in an event-data memory store (M5 in Figure 7; Table 7 on page 34).

Table 16 describes the output variables returned from read_evloc_db_tables() to *main()*. The num_events variable is 0 if no origin records were retrieved from the **origin** table. Otherwise, num_events is the number of events retrieved from the **origin** table for the time segment being processed.

TABLE 16: OUTPUT VARIABLES FROM READ_EVLOC_DB_TABLES()

Type	Variable Name	Use	Description
`int`	*num_events*	R	number of events read from the input **origin** database table
`Site **`	*sites*	A	pointer to array of `Site` database table structures
`int *`	*num_sites*	A	number of elements in array of `Site` database table structures
`Ev **`	*ev_anch*	A	pointer to first element in `Ev` linked list

Interfaces

Only `main()` calls `read_evloc_db_tables()` within *EvLoc*. `read_evloc_db_tables()` calls several lower-level *EvLoc* processing units (see Table 12 on page 50).

`read_evloc_db_tables()` also calls processing units from several global libraries (Table 12). It calls the *libmagnitude* processing units `setup_mag_facilities()` to read the earth-model files, `build_mag_obj()` to store event and magnitude data in internal memory, and `copy_magnitudes()` to copy magnitude data from one `Magnitude` object (Table 54 on page 133) to another. `read_evloc_db_tables()` also calls the *libloc* processing units `setup_tt_facilities()` to read travel-time tables and `read_sasc()` to read slowness/azimuth station correction data [IDC-7.1.5].

`read_evloc_db_tables()` also calls multiple *libgdi* processing units. It initializes the GDI for dynamic linking to the input database and error handling by calls to `gdi_init()` and `gdi_error_init()`, respectively. It calls `gdi_open()` to establish a connection to the input database account. `read_evloc_db_tables()` retrieves all records from database tables using `gdi_get_ArrayStructs()`. If problems occur during any interaction with the database, then `gdi_error_get()` returns information about the error to `read_evloc_db_` `tables()`. Finally, `read_evloc_db_tables()` calls `gdi_close()` to close the database connection. Refer to the *libgdi* man page for descriptions of the interfaces.

Error States

`read_evloc_db_tables()` interprets the status codes or return values returned from all lower-level *EvLoc* and global library processing units that it calls (see Table 12 on page 50 for hierarchy). OK status codes or acceptable return values indicate to `read_evloc_db_tables()` that processing should continue. Error status codes or out-of-bounds return values indicate to `read_evloc_db_tables()` that it should terminate all further *EvLoc* processing and write an error message to stderr.

`read_evloc_db_tables()` checks for memory allocation errors for each element of the `Ev` linked list. If a memory allocation error occurs, then `read_evloc_db_tables()` writes an error message to stderr and terminates all further *EvLoc* processing.

write_evloc_db_tables()

`write_evloc_db_tables()` writes station- and network-magnitude data to an output database.

Input/Processing/Output

`write_evloc_db_tables()` is called by *main()* as part of the *Obtain Updated Magnitude Results* process (1.4 in Figure 7 on page 32). Table 17 describes the input variables to `write_evloc_db_tables()`. The data source for the *evloc_par* input variable is the station, event, and control-parameter data memory store (M6 in Figure 8 on page 37; Table 5 on page 33). The data source for the *ev_anch* input variable is the event-data memory store (M5 in Figure 7; Table 7 on page 34).

TABLE 17: INPUT VARIABLES TO WRITE_EVLOC_DB_TABLES()

Type	Variable Name	Use	Description
`EvLoc_Par`	*evloc_par*	A	`EvLoc_Par` structure
`Ev **`	*ev_anch*	A	pointer to first element in `Ev` linked list

`write_evloc_db_tables()` retrieves station- and network-magnitude data from memory and writes the data to an output database account (database store Db2 in Figure 7). The input and output database accounts may be the same. `write_evloc_db_tables()` uses the GDI to establish a connection to the **output** database, retrieve a new set of magnitude identifiers (*magids*) from the **lastid** table, clean old records from the `input` database account if the input and output accounts are identical, write records to the database, close the connection, and handle any errors that occur during these processes.

`write_evloc_db_tables()` inserts the station- and network-magnitude data from the `Stamag` and `Netmag` database table structures (see Table 2 on page 14), respectively, into the output **stamag** and **netmag** database tables. These structures are nested within the `Magnitude` objects (Table 54 on page 133) that are themselves nested within the `Mag_ptr` structures (Table 53 on page 132) of each element in the `Ev` linked list (Table 52 on page 130). `write_evloc_db_tables()` assigns a unique *magid* to `Stamag` and `Netmag` elements for each distinct event and magtype combination successfully computed.

`write_evloc_db_tables()` inserts the event data from the `Origin` database table structures (Table 2) nested within each element of the `Ev` linked list into the output **origin** database table. The data in the `Origin` structures includes network magnitudes estimated for the processed events. These data may also include associated *magids* from the `Stamag` and `Netmag` structures written to the magnitude identifier members (`mbid`, `msid`, and `mlid`) of the `Origin` structures.

`write_evloc_db_tables()` also optionally inserts the magnitude control data from the `Event_control` database table structure (Table 2) within each element of the `Ev` linked list into the output **event_control** table.

The only output of `write_evloc_db_tables()` is a status code that is returned to *main()*.

Interfaces

Only `main()` and `make_predicts()` call `write_evloc_db_tables()` (see Table 12 on page 50). `write_evloc_db_tables()` does not call any lower-level *EvLoc* processing units.

`write_evloc_db_tables()` calls processing units from several global libraries (Table 12). It calls the *libstdtime* processing unit `stdtime_get_lddate()` to compute an *lddate*, which is inserted in all database table structures whose contents are written to the output database tables (Refer to the *libstdtime* man page for a description of this interface).

`write_evloc_db_tables()` also calls multiple *libgdi* processing units. It initializes the GDI for dynamic linking to the output database and error handling via calls to `gdi_init()` and `gdi_error_init()`, respectively. It calls `gdi_open()` to establish a connection to the output database account. `write_evloc_db_tables()` retrieves *magids* from the **lastid** table using `gdi_get_counter()`. It deletes old records from the **origin**, **stamag**, **netmag**, and **event_control** tables in the input database account using `gdi_submit()`. `write_evloc_db_tables()` calls `gdi_add_ArrayStructs()` to write records to the output database tables. If problems occur during any interaction with the database, then `gdi_error_get()` returns information about the error to `write_evloc_db_tables()`, and `gdi_rollback()` reconstructs the initial state of the database account. If no problems occur, then `gdi_commit()` commits the new records to the database. Finally, `write_evloc_db_tables()` calls `gdi_close()` to close the database connection. (Refer to the *libgdi* man page for descriptions of the interfaces.)

Error States

`write_evloc_db_tables()` interprets the status codes or return values returned from all lower-level *EvLoc* and global library processing units that it calls (see Table 12 on page 50 for hierarchy). OK status codes or acceptable return values indicate to `write_evloc_db_tables()` that processing should continue. Error status codes or out-of-bounds return values indicate to `write_evloc_db_tables()` that it should terminate all further *EvLoc* processing and write an error message to stderr.

LIBMAGNITUDE PROCESSING UNITS

The *libmagnitude* portion of the Event Magnitude software includes 48 processing units. For the remainder of this chapter, the term "processing unit" describes a particular *libmagnitude* entity. The role of each processing unit is identified more specifically by the terms "interface" and "function." An interface is a processing unit that exchanges data between an external application and one or more internal *libmagnitude* processing units (external interface), or exchanges data between two internal processing units (internal interface). A function is a processing unit that is called by an interface or another function to perform a specific task or set of tasks.

The 48 *libmagnitude* processing units are listed in Tables 18 through 22. Tables 18 through 21 list the hierarchy of the 34 *libmagnitude* processing units that are associated with each of the four processes shown in Figure 10 on page 43. Table 22 lists a number of additional utility functions provided in *libmagnitude*. Tables 18 through 21 are referred to as hierarchy tables, because they identify the hierarchy of each *libmagnitude* processing unit with respect to applications and other *libmagnitude* processing units within a particular process. These tables list all applications and processing units that call and are called by each processing unit. Their structure is governed by control flow: external interfaces are identified first, followed by any internal *libmagnitude* interfaces, and concluding with internal *libmagnitude* functions. The internal interfaces and functions are listed in the order they are called by a higher-level processing unit.

TABLE 18: HIERARCHY OF PROCESSING UNITS IN READ EARTH-MODEL DATA PROCESS

Processing Unit	Description	Called from	Calls to
`setup_mag_facilities()`	external interface; interfaces between applications and *libmagnitude* processing units that read and store data from earth-model files	applications (for example, *StaPro*, GA, *WaveExpert*, *EvLoc*, and *ARS*)	`setup_mc_tables()`, `set_sta_TL_pt()`, `mag_error_msg()`
`setup_mc_tables()`	internal interface; interfaces between `setup_mag_facilities()` and functions that read and store data from earth-model files	`setup_mag_facilities()`	`read_mdf()`, `mag_error_msg()`, `read_tlsf()`, `TL_error_msg()`
`set_sta_TL_pt()`	function; links stations to TLM description data	`setup_mag_facilities()`	none
`mag_error_msg()`	function; links magnitude status code with status message string	`setup_mag_facilities()`, `setup_mc_tables()`	none
`read_mdf()`	function; reads magnitude specification and bulk-station-correction data from single MDF; stores data in earth-model-data memory store (Table 8 on page 38)	`setup_mc_tables()`	none

TABLE 18: HIERARCHY OF PROCESSING UNITS IN READ EARTH-MODEL DATA PROCESS (CONTINUED)

Processing Unit	Description	Called from	Calls to
`read_tlsf()`	internal interface and function; reads TLM pathway and TLM description data from single TLSF and stores data in earth-model-data memory store (Table 8 on page 38)	`setup_mc_tables()`	`free_tl_table()`, `read_tl_table()`
`TL_error_msg()`	function; links transmission-loss status code with status message string	`setup_mc_tables()`	none
`free_tl_table()`	function; frees memory allocated to TLMs	`read_tlsf()`	none
`read_tl_table()`	function; reads transmission-loss (magnitude correction) and modeling error data from single TLM and stores data in earth-model-data memory store (Table 8 on page 38)	`read_tlsf()`	none

TABLE 19: HIERARCHY OF PROCESSING UNITS IN BUILD MAGNITUDE DATA STORE PROCESS

Processing Unit	Description	Called from	Calls to
`build_mag_obj()`	external interface and function; stores event and magnitude specification data in `Magnitude` objects (see Table 11 on page 45)	applications (for example, *EvLoc* and *ARS*)	`get_magtype_features()`, `get_delta_for_sta()`, `valid_phase_for_TLtype()`, `valid_range_for_TLtable()`
`copy_magnitudes()`	external interface and function; copies `Magnitude` objects (see Table 11 on page 45)	applications (for example, *EvLoc*)	none
`free_magnitudes()`	external interface and function; frees memory allocated to `Magnitude` objects (see Table 11 on page 45)	applications (for example, *ARS*)	none
`get_magtype_features()`	external interface and function; stores magnitude specification data in `Magnitude` objects (see Table 11 on page 45)	applications (for example, *ARS*), *build_mag_obj()*	none
`valid_phase_for_TLtype()`	external interface, internal interface, and function; determines whether phase is valid for given TLtype	applications (for example, *ARS*), `build_mag_obj()`	`get_TLMD_index()`
`valid_range_for_TLtable()`	internal interface and function; determines whether distance and depth are valid for given TLM	`build_mag_obj()`	`get_TL_indexes()`

TABLE 19: HIERARCHY OF PROCESSING UNITS IN BUILD MAGNITUDE DATA STORE PROCESS (CONTINUED)

Processing Unit	Description	Called from	Calls to
`get_delta_for_sta()`[1]	function; computes event-to-station distance	`build_mag_obj()`, `calc_mags()`	*libgeog* function `dist_azimuth()`
`get_TLMD_index()`	function; determines index associated with TLM for given TLtype	`valid_phase_for_TLtype()`	none
`get_TL_indexes()`[2]	function; determines indexes associated with all lines in TLSF and TLM for given TLtype, station, phase, and channel	`get_mag_indexes()`, `valid_range_for_TLtable()`	none

1. This processing unit is also called in the Estimate Network-magnitude Data process.

2. This processing unit is also called in the Estimate Station-magnitude Data process.

TABLE 20: HIERARCHY OF PROCESSING UNITS IN ESTIMATE STATION-MAGNITUDE DATA PROCESS

Processing Unit	Description	Called from	Calls to
`station_magnitude()`	external or internal interface and function; estimates station-magnitude data	applications (*StaPro*, GA, and *WaveExpert*), `calc_mags()`,[1] `abbrev_sta_mag()`	`initialize_sm_info()`, `get_mag_indexes()`, `interp_for_tl_value()`, `get_TL_ts_corr()`, `get_tl_model_error()`, `get_meas_error()`
`get_delta_for_sta()`[2]	function; computes event-to-station distance	`build_mag_obj()`, `calc_mags()`[1]	*libgeog* function, `dist_azimuth()`

TABLE 20: HIERARCHY OF PROCESSING UNITS IN ESTIMATE STATION-MAGNITUDE DATA PROCESS (CONTINUED)

Processing Unit	Description	Called from	Calls to
`initialize_sm_info()`	function; initializes station-magnitude-data memory store (Table 9 on page 40) to default values	`station_magnitude()`	none
`get_mag_indexes()`	internal interface and function; determines indices associated with all lines in MDF and TLSF for given magtype, station, phase, and channel	`station_magnitude()`	`get_TL_indexes()`
`interp_for_tl_value()`	function; retrieves distance/depth-dependent magnitude correction from earth-model-data memory store (Table 8 on page 38)	`station_magnitude()`	*libinterp* function, `interpolate_table_value()`
`get_TL_ts_corr()`[3]	function; retrieves test-site correction from earth-model-data memory store (Table 8 on page 38)	`station_magnitude()`	none
`get_tl_model_error()`	function; retrieves modeling error from earth-model-data memory store (Table 8 on page 38)	`station_magnitude()`	none
`get_meas_error()`	function; designed to estimate measurement error for given snr	`station_magnitude()`	none

TABLE 20: HIERARCHY OF PROCESSING UNITS IN ESTIMATE STATION-MAGNITUDE DATA PROCESS (CONTINUED)

Processing Unit	Description	Called from	Calls to
`get_TL_indexes()`[2]	function; determines indexes associated with all lines in TLSF and TLM for given TLtype, station, phase, and channel	`get_mag_indexes()`, `valid_range_for_TLtable()`	none
`abbrev_sta_mag()`	external interface (obsolete); similar to `station_magnitude()`, but returns less data to application	none	`station_magnitude()`

1. This processing unit is considered to be part of the Estimate Network-magnitude Data process, even though a portion of this processing unit is part of the Estimate Station-magnitude Data process.
2. This processing unit is also called in the Build Magnitude Data Store process.
3. This processing unit is not applicable to the IDC.

TABLE 21: HIERARCHY OF PROCESSING UNITS IN ESTIMATE NETWORK-MAGNITUDE DATA PROCESS

Processing Unit	Description	Called from	Calls to
`calc_mags()`	external interface and function; estimates network magnitudes and uncertainties for set of magtypes	applications (for example, *EvLoc* and *ARS*)	`get_delta_for_sta()`,[1] `station_magnitude()`,[2] `network_mag()`, `mag_boot_strap()`, `mag_get_compute_upper_bounds()`

TABLE 21: HIERARCHY OF PROCESSING UNITS IN ESTIMATE NETWORK-MAGNITUDE DATA PROCESS (CONTINUED)

Processing Unit	Description	Called from	Calls to
`initialize_mag_params()`	external interface and function; initializes magnitude control parameters (see Table 10 on page 44) to default values	applications (for example, *EvLoc*)	none
`mag_set_compute_upper_bounds()`	external interface and function; sets flag indicating whether or not to estimate upper-bound magnitudes and uncertainties	applications (for example, *EvLoc* and *ARS*)	none
`network_mag()`	internal interface and function; estimates network-average magnitude, standard deviation, and uncertainty	`calc_mags()`	`only_bound_amps()`, `mag_max_lik()`
`mag_boot_strap()`	internal interface and function; estimates MLE magnitude, standard deviation, and two uncertainties using bootstrap resampling	`calc_mags()`	`mag_max_lik()`
`mag_get_compute_upper_bounds()`	function; retrieves flag indicating whether or not to estimate upper-bound magnitudes and uncertainties	`calc_mags()`	none

TABLE 21: HIERARCHY OF PROCESSING UNITS IN ESTIMATE NETWORK-MAGNITUDE DATA PROCESS (CONTINUED)

Processing Unit	Description	Called from	Calls to
`only_bound_amps()`	function; estimates upper- or lower-bound magnitude and standard deviation.a	`network_mag()`	none
`mag_max_lik()`	function; estimates MLE magnitude and standard deviation.a	`network_mag()`, `mag_boot_strap()`	none

1. This processing unit is also called in the Build Magnitude Data Store process.
2. This processing unit is considered to be part of the Estimate Station-magnitude Data process.

Table 22 lists the 14 *libmagnitude* processing units that are self-contained. Most of them are not called by any application or other processing unit and are generally not described outside of Table 22. However, calls to any of them could be inserted into applications or other *libmagnitude* processing units in the future.

TABLE 22: STAND-ALONE PROCESSING UNITS

Processing Unit	Description	Called from	Calls to
`reset_algorithm()`	external interface and function; changes magnitude algorithm code for given magtype	none	none
`reset_amptypes()`	external interface and function; changes arrival-based and origin-based amptype identifiers for given magtype	none	none
`reset_sd_baseline()`	external interface and function; changes baseline uncertainty for given magtype	none	none

TABLE 22: STAND-ALONE PROCESSING UNITS (CONTINUED)

Processing Unit	Description	Called from	Calls to
`reset_sd_limits()`	external interface and function; changes uncertainty boundaries for given magtype	none	none
`reset_wgt_ave_flag()`	external interface and function; changes weighted average flag for given magtype	none	none
`reset_max_dist()`	external interface and function; changes maximum distance for a magtype	applications (for example, *EvLoc* and *ARS*)	none
`reset_min_dist()`	external interface and function; changes minimum distance for a magtype	applications (for example, *EvLoc* and *ARS*)	none
`revert_algorithm()`	external interface and function; reverts to magnitude algorithm code specified in magnitude description data (see Table 8 on page 38) for given magtype	none	none
`revert_amptypes()`	external interface and function; reverts to arrival-based and origin-based amptype identifiers specified in magnitude description data (see Table 8 on page 38) for given magtype	none	none
`revert_sd_baseline()`	external interface and function; reverts to baseline uncertainty specified in magnitude description data (see Table 8 on page 38) for given magtype	none	none
`revert_sd_limits()`	external interface and function; reverts to uncertainty boundaries specified in magnitude description data (see Table 8 on page 38) for given magtype	none	none

TABLE 22: STAND-ALONE PROCESSING UNITS (CONTINUED)

Processing Unit	Description	Called from	Calls to
`revert_wgt_ave_flag()`	external interface and function; reverts to setting of weighted average flag specified in magnitude description data (see Table 8 on page 38) for given magtype	none	none
`revert_max_dist()`	external interface and function; reverts to maximum distance specified in magnitude description data (see Table 8 on page 38) for given magtype	none	none
`revert_min_dist()`	external interface and function; reverts to minimum distance specified in magnitude description data (see Table 8 on page 38) for given magtype	none	none

Eleven processing units that are critical interface/functional components of the four *libmagnitude* processes are listed below. The following paragraphs describe the design of these units, including any constraints or unusual features in the design. The logic of the software and any applicable procedural commands are also provided. Design details of the remaining 37 processing units are not discussed outside of Tables 18 through 22, because they are interfaces or functions that perform simple or low-level specialized tasks or sets of tasks.

- *Read Earth-model Data* process
  - `setup_mag_facilities()`
  - `read_mdf()`
  - `read_tlsf()`
  - `read_tl_table()`
- *Build Magnitude Data Store* process
  - `build_mag_obj()`

- *Estimate Station-magnitude Data* process
  - `station_magnitude()`
- *Estimate Network-magnitude Data* process
  - `calc_mags()`
  - `network_mag()`
  - `mag_boot_strap()`
  - `mag_max_lik()`
  - `only_bound_amps()`

The description of each processing unit includes a table that describes its input and output variables. Each table lists the name and data type, and describes each input or output variable. In addition, the Use column assigns a classification to each variable. The Use column is "A" (argument) if the variable is input or output through the processing unit's argument list, "M" (memory) if the variable is input or output by reading to or writing from a static memory store, and it is "R" (return) if the variable is the return value from the processing unit.

setup_mag_facilities()

`setup_mag_facilities()` is the external interface between applications and *libmagnitude* processing units that read the earth-model files and store the earth-model data in internal memory structures.

Input/Processing/Output

`setup_mag_facilities()` is a core element of both the station- and network-magnitude modes of operation. The external interface is called by an application operating in either mode to initiate the *Read Earth-model Data* process (2.1 in Figure 10 on page 43). Table 23 describes the input variables to `setup_mag_facilities()`.

TABLE 23: INPUT VARIABLES TO SETUP_MAG_FACILITIES()

Type	Variable Name	Use	Description
`char *`	*tl_model_filename*	A	TLSF pathname
`char *`	*mdf_filename*	A	MDF pathname
`char **`	*list_of_magtypes*	A	list of magtypes to be estimated
`int`	*num_magtypes*	A	number of elements in list of magtypes
`Site *`	*sites*	A	array of `Site` database table structures
`int`	*num_sites*	A	number of elements in array of `Site` structures

The only output of `setup_mag_facilities()` is an integer status code that is returned to the calling application.

Interfaces

All external applications that operate in either station- or network-magnitude mode and require data from the earth-model files must call *libmagnitude* through `setup_mag_facilities()`, as indicated in Figure 8 on page 37 and Figure 10 on page 43. *StaPro*, the GA Subsystem (including *GAcons*), *WaveExpert*, *EvLoc*, and *ARS* all call `setup_mag_facilities()`. `setup_mag_facilities()` only needs to be called once by an application.

`setup_mag_facilities()` calls the lower-level *libmagnitude* processing units listed in Table 18 on page 63.

Error States

`setup_mag_facilities()` is an external interface and is designed to be called directly by an external application. As a result, the *mdf_filename* and *tl_model_file-name* input arguments (Table 23) are checked within the lower-level functions read_mdf() and read_tlsf(), respectively, to ensure that the MDF and TLSF are located on the filesystem as specified. The application is responsible for ensuring that the remaining input arguments contain valid data.

setup_mag_facilities(), setup_mc_tables(), set_sta_TL_pt(), read_mdf(), read_tlsf(), and read_tl_table() return integer status codes to the next higher-level interface (see Table 18 for hierarchy). The higher-level interfaces interpret the status code and determine whether or not to continue processing. Tables 24 and 25 describe these status codes, which are associated with reading, linking, and storing data from the magnitude and transmission-loss files, respectively. The Status Type column is either "OK," "error," or "warning." An OK status type indicates to the higher-level interface that it should continue its processing because processing in the lower-level function was successful. A returned error code tells the higher-level interface to terminate all further processing. A warning status code indicates to the higher-level interface that part of its functionality should be skipped. The Status Message String column specifies the message string associated with the status code. setup_mag_facilities() and setup_mc_tables() call mag_error_msg() and TL_error_msg() to write the message string to stderr if error codes are returned to them from set_sta_TL_pt(), read_mdf(), or read_tlsf(). The Source Processing Unit(s) column lists the processing unit(s) that may report a specific status code condition.

TABLE 24: MAGNITUDE STATUS CODES

Status Code	Status Type	Status Message String	Source Processing Unit(s)
0	OK	Magnitude: Successful magnitude computed!	setup_mag_facilities(), setup_mc_tables(), set_sta_TL_pt(), read_mdf()
1	error	MDreadErr1: Cannot open MDF!	read_mdf()
2	error	MDreadErr2: MDF incorrectly formatted!	read_mdf()
3	not used	n/a	n/a

TABLE 24: MAGNITUDE STATUS CODES (CONTINUED)

Status Code	Status Type	Status Message String	Source Processing Unit(s)
4	error	MDreadErr4: Error allocating memory while reading mag info!	read_mdf()
5	error	SSgetErr1: No input site table info available for Sta_Pt!	set_sta_TL_pt()
6	error	SSgetErr2: Error allocating memory while trying to set Sta_Pt info!	set_sta_TL_pt()
7	not used	n/a	n/a

TABLE 25: TRANSMISSION-LOSS STATUS CODES

Status Code	Status Type	Status Message String	Source Processing Unit(s)
0	OK	TL: Successful TL condition!	setup_mag_facilities(), setup_mc_tables(), read_tlsf(), read_tl_table()
1	warning	TLreadWarn1: A requested TL file was not found!	read_tl_table()
2	error	TLreadErr1: Cannot open TLSF!	read_tlsf()
3	error	TLreadErr2: TLSF incorrectly formatted!	read_tlsf()
4	error	TLreadErr3: No TL tables could be found!	read_tlsf()
5	error	TLreadErr4: TL table incorrectly formatted!	read_tl_table()
6	error	TLreadErr5: TL modelling error table incorrectly formatted!	read_tl_table()

TABLE 25: TRANSMISSION-LOSS STATUS CODES (CONTINUED)

Status Code	Status Type	Status Message String	Source Processing Unit(s)
7[1]	error	`TLreadErr6: TL test-site corr. file incorrectly formatted!`	`read_tl_table()`
8	error	`TLreadErr7: Error allocating memory while reading TL info!`	`read_tlsf()`, `read_tl_table()`

1. This status code is not applicable to the IDC.

`set_sta_TL_pt()`, `read_mdf()`, `read_tlsf()`, and `read_tl_table()` write an additional, more detailed message string to stderr if they encounter one of the error or warning conditions listed in Tables 24 and 25.

OK status codes are passed upward through the processing unit hierarchy and are ultimately returned to the calling application by `setup_mag_facilities()`, unless overridden by an error code. Error status codes are also passed up the processing unit hierarchy and are returned to the application. Warning status codes are only passed upwards one level in the hierarchy and are replaced by either OK or error status codes within the higher-level interface, depending on whether or not the interface encounters an error condition.

read_mdf()

`read_mdf()` is a function that reads data from an MDF and records the data in a memory store.

Input/Processing/Output

`read_mdf()` is a function that is called by `setup_mc_tables()` as part of the *Read Earth-model Data* process. See Figure 8 on page 37 and Figure 10 on page 43 for the relationship of this process to applications and other *libmagnitude* processes. See Figure 9 on page 41 and Table 18 on page 63 for the relationship between this function and other *libmagnitude* functions within the *Read Earth-model Data* process.

Table 26 describes the input variables to `read_mdf()`. The data source for these variables is the station, event, and control-parameter data memory store (M6 in Figures 8, 9, and 10; Table 10 on page 44).

TABLE 26: INPUT VARIABLES TO READ_MDF()

Type	Variable Name	Use	Description
`char *`	*mdf_filename*	A	MDF pathname
`char **`	*list_of_magtypes*	A	list of magtypes to be estimated
`int`	*num_req_magtypes*	A	number of elements in list of magtypes

`read_mdf()` reads the MDF specified in the input variable *mdf_filename*. The MDF (D2.a in Figure 9) is composed of two distinct sections. The first section contains magnitude description (or magnitude control) data, and the second section contains bulk-station-correction data.

The magnitude description data are lines of magnitude control settings for a specific magtype. `read_mdf()` parses the magnitude description data and stores each line of the control settings in an element of an array of `Mag_Descrip` structures (Table 55 on page 134). Only the magnitude control settings associated with the magtypes specified in the input variable *list_of_magtypes* are stored in the array of `Mag_Descrip` structures.

The values stored in the `Mag_Descrip` structure members `det_amptype`, `ev_amptype`, `algo_code`, `dist_min`, `dist_max`, `sglim1`, `sglim2`, `sgbase`, and `apply_wgt` may be changed by the application after control returns to the application from `setup_mag_facilities()`. These original control settings are stored in the analogously named `Mag_Descrip` component members prefixed with `orig_`. To change the value of one of the modifiable members, the application calls one of the external interfaces prefixed with *reset_* (Table 22 on page 70).

After the value of one of these structure members has been changed, the original control settings may be recovered by the application from the orig_ members through a call to the relevant interface prefixed with *revert_* (Table 22).

read_mdf() identifies the unique TLtypes stored in the TLtype members of Mag_Descrip. These unique TLtypes are copied into a linked list of TL_Pt structures. This linked list is used in read_tlsf() to define which TLSF data will be stored in memory.

The second section of the MDF contains bulk-station-correction data. The bulk-station-correction data are lines in the MDF of static magnitude corrections and errors given for specific station and TLtype combinations. read_mdf() parses the bulk-station-correction data and stores each line of data in an element of an array of Mag_Sta_TLType structures (Table 56 on page 135). Any default bulk station corrections and uncertainties associated with a particular TLtype are also stored in the def_mag_corr and def_mag_corr_err members of the Mag_Descrip structure.

read_mdf() returns the output variables listed in Table 27 to setup_mc_tables(). All variables other than *icode* are stored as components of the earth-model-data memory store.

TABLE 27: OUTPUT VARIABLES FROM READ_MDF()

Data Type	Variable Name	Use	Description
int	*icode*	R	status code
Mag_Descrip **	*mag_descrip_ptr*	A	pointer to array of Mag_Descrip structures
int *	*num_md*	A	number of elements in array of Mag_Descrip structures
Mag_Sta_TLType **	*mag_sta_tltype_ptr*	A	pointer to array of Mag_Sta_TLType structures
int *	*num_mst*	A	number of elements in array of Mag_Sta_TLType structures
TL_Pt **	*list_of_TLtypes_ptr*	A	pointer to linked list of TL_Pt structures

Interfaces

Only `setup_mc_tables()` calls `read_mdf()` (see Table 18 on page 63 for hierarchy). `read_mdf()` does not call any lower-level *libmagnitude* processing units.

Error States

`read_mdf()` is a function designed not to be called directly by an external application. As a result, most of the input arguments in Table 26 are not checked within `read_mdf()` to ensure valid content. An exception is `mdf_filename`, which is checked for valid content.

`read_mdf()` writes an error message to stderr, terminates all further processing within itself, and returns the appropriate error status code (Table 24 on page 75) to *setup_mc_tables()* if it encounters an error condition. If `read_mdf()` reads, parses, and stores the data from the MDF without encountering any error conditions, then it returns an OK status code to *setup_mc_tables()*.

read_tlsf()

`read_tlsf()` is a function that reads data from a TLSF and stores the data in a memory store.

Input/Processing/Output

`read_tlsf()` is a function that is called by `setup_mag_facilities()` as part of the *Read Earth-model Data* process. See Figures 8 on page 37 and 10 on page 43 for the relationship of this process to applications and other *libmagnitude* processes. See Figure 9 on page 41 and Table 18 on page 63 for the relationship between this function and other *libmagnitude* functions within the *Read Earth-model Data* process.

Table 28 describes the input variables to `read_tlsf()`. The data source for the *list_of_TLtypes* variable is the earth-model-data memory store (M1 in Figure 9; Table 8 on page 38). The data source for the remaining variables is the station, event, and control-parameter data memory store (M6 in Figures 8, 9, and 10; Table 10 on page 44).

TABLE 28: INPUT VARIABLES TO READ_TLSF()

Type	Variable Name	Use	Description
`char *`	*tl_model_filename*	A	TLSF pathname
`TL_Pt *`	*list_of_TLtypes*	A	pointer to linked list of `TL_Pt` structures
`Site *`	*sites*	A	array of `Site` database table structures
`int`	*num_sites*	A	number of elements in array of `Site` structures

`read_tlsf()` reads the TLSF as specified in the input variable *tl_model_filename*. The TLSF (D2.b in Figure 9) is composed of three distinct sections. The first section contains TLM pathway data, the second section contains default TLM description data, and the third section contains station-specific TLM description data. Each line of the file specifies a particular element of this data.

The TLM pathway data are lines of TLM root names and pathways that specify the directory locations of all TLMs relative to the directory location of the TLSF. `read_tlsf()` parses the TLM pathway data and stores each line in an element of an array of `TL_Model_Path` structures (Table 63 on page 141).

The second section of the TLSF contains default TLM description data. The default TLM description data are lines of default TLM root names and phase data given for specific TLtypes. The default TLM description data link TLtypes and phase names to default TLMs. `read_tlsf()` parses the default TLM description data and stores each line of data in an element of an array of `TLType_Model_Descrip` structures (Table 61 on page 139). Default TLM description data are used in the absence of any station-specific data.

The third section of the TLSF contains optional station-specific TLM description data. The station-specific TLM description data are lines of station-specific root names and phase and channel data given for specific station and TLtype combinations. The station-specific TLM description data link stations, TLtypes, phase names, and channel/frequency identifiers to station-specific TLMs. `read_tlsf()` parses the station-specific TLM description data and stores each line of data in an element of an array of `Sta_TL_Model` structures (Table 60 on page 138). Station-specific TLM description data override default data for a given station and TLtype combination.

`read_tlsf()` only stores the relevant data from these three TLSF sections in internal memory. Only default TLM description data associated with the TLtypes listed in the *list_of_TLtypes* input variable are stored in the array of `TLType_Model_Descrip` structures. Only station-specific TLM description data associated with the stations and TLtypes listed in the *Site* and *list_of_TLtypes* input variables are stored in the array of `Sta_TL_Model` structures. In addition, only the TLM pathway data with root names that are identical to default root names from the default TLM description data are stored in the `TL_Model_Path` structures.

`read_tlsf()` calls `read_tl_table()` to read a single TLM and stores its data in internal memory. `read_tl_table()` is called multiple times if multiple TLMs are read. The default and station-specific TLM filenames are constructed in `read_tl_table()` from the TLM pathway data and portions of the (default or station-specific) TLM description data. After `read_tl_table()` reads a TLM, it stores the data in a `TL_Table` structure (Table 64 on page 141). Each time `read_tl_table()` is called, it returns a pointer to the `TL_Table` structure to `read_tlsf()`. `read_tlsf()` stores each pointer in an element of an array of pointers to `TL_Table` structures.

`read_tlsf()` populates the output variables listed in Table 29. The *icode* is returned to `setup_mag_facilities()`. The remaining variables are stored in the earth-model-data memory store (M1 in Figures 8, 9, and 10; Table 8 on page 38).

TABLE 29: OUTPUT VARIABLES FROM READ_TLSF()

Type	Variable Name	Use	Description
`int`	*icode*	R	status code
`TL_Model_Path *`	*tl_model_path*	M	pointer to an array of `TL_Model_Path` structures
`int`	*num_TL_models*	M	number of elements in the `TL_Model_Path` structures
`TLType_Model_Descrip *`	*tltype_model_descrip*	M	pointer to an array of `TLType_Model_Descrip` structures
`int`	*num_TLMD*	M	number of elements in the `TLType_Model_Descrip` structures
`Sta_TL_Model *`	*sta_tl_model*	M	pointer to an array of `Sta_TL_Model` structures
`int`	*num_STM*	M	number of elements in the `Sta_TL_Model` structures
`TL_Table **`	*tl_table_ptr*	M	array of pointers to `TL_Table` structures
`int`	*num_TL_tables*	M	number of elements in the array of pointers to `TL_Table` structures

Interfaces

Only *setup_mc_tables()* calls `read_tlsf()` (see Table 18 on page 63 for hierarchy). `read_tlsf()` calls the lower-level *libmagnitude* processing units `read_tl_table()` and `free_tl_table()`.

Error States

`read_tlsf()` is a function designed not to be called directly by external applications. As a result, most of the input arguments in Table 28 are not checked within `read_tlsf()` to ensure valid content. An exception is *tl_model_filename*, which is checked for valid content.

`read_tlsf()` interprets the status code returned from `read_tl_table()` (see Table 18 for hierarchy). Table 25 on page 76 lists the status codes that `read_tl_table()` may return. An OK status code indicates to `read_tlsf()` that processing was successful. Error status codes indicate to `read_tlsf()` that it should terminate all further processing and return the same error code to `setup_mc_tables()`. A warning status code indicates to `read_tlsf()` that the TLM file was not found, so it should skip any further processing associated with this TLM.

`read_tlsf()` writes an error message to stderr, terminates all further processing within itself, and returns the appropriate error status code (Table 25) to `setup_mc_tables()` if it encounters an error condition. If `read_tlsf()` reads, parses, and stores the data from the TLSF without encountering any error conditions, then it returns an OK status code to `setup_mc_tables()`.

`read_tlsf()` checks for warning conditions caused by reading redundant, duplicate, or unusable station-specific TLM description data. If a station-specific TLM description is identical to a default TLM description, identical to another station-specific TLM description, or contains a TLtype that is not specified in the default TLM description data, then the station-specific TLM description data are redundant, duplicated, or unusable, respectively. `read_tlsf()` ignores this station-specific TLM description data, writes a warning message to stderr, and continues its processing. The warning message is of the form:

```
read_tlsf: Warning! STM: <STA>/<TLTYPE>/<TLM ROOT NAME>
<SUMMARY>
STM line: <S-S TLM DATA>
will be ignored!
```

where *<STA>* is the station name, *<TLTYPE>* is the TLtype, and *<TLM ROOT NAME>* is the root name associated with the station-specific TLM description data. *<S-S TLM DATA>* is an entire line of station-specific TLM description data, including any phase or channel/frequency dependencies. *<SUMMARY>* is a summary of the warning condition encountered. Table 30 lists the summaries for the three warning conditions.

TABLE 30: READ_TLSF() WARNING SUMMARIES

Warning Condition	Summary
redundant	`found to be redundant with info specified in tltype_model_descrip!`
duplicate	`found to be a duplicate with another STM record!`
unusable	`not associated with any tltype_model_descrip TLtype definition!`

read_tl_table()

`read_tl_table()` is a function that reads data from a TLM and stores the data in a memory store.

Input/Processing/Output

`read_tl_table()` is a function that is called by `read_tlsf()` as part of the *Read Earth-model Data* process. See Figure 8 on page 37 and Figure 10 on page 43 for the relationship of this process to applications and other *libmagnitude* processes. See Figure 9 on page 41 and Table 18 on page 63 for the relationship between this function and other *libmagnitude* functions within the *Read Earth-model Data* process.

Table 31 describes the input variables to `read_tl_table()`. The data source for these variables is the earth-model-data memory store (M1 in Figure 9; Table 8 on page 38). The *tl_model* variable may contain the root name of a default or station-specific TLM.

TABLE 31: INPUT VARIABLES TO READ_TL_TABLE()

Type	Variable Name	Use	Description
char *	*dir_pathway*	A	directory location of TLM relative to TLSF
char *	*TLtype*	A	transmission-loss descriptor
char *	*tl_model*	A	TLM root name
char *	*phase*	A	phase name
char *	*chan*	A	channel identifier

`read_tl_table()` constructs the full TLM filename. The TLM filename is formatted from the input variables in Table 31 as follows:

```
dir_pathway/tl_model.TLtype.phase.chan
```

where the `phase` and `chan` suffixes are optional. These suffixes are only used if a default or station-specific TLM is phase dependent or if a station-specific TLM is channel/frequency-dependent. The `phase` and `chan` suffixes are only added to the filename if they are character strings other than "`-`".

`read_tl_table()` reads the default or station-specific TLM whose filename was constructed above. A TLM (D2.c in Figure 9) is composed of two distinct sections. The first section contains transmission-loss data, and the second section contains transmission-loss modeling error data.

The transmission-loss (in the seismic case, magnitude correction) data are distance/depth-dependent estimates of transmission loss. `read_tl_table()` parses the distance and depth samples for which the transmission-loss values are estimated and then parses the magnitude corrections themselves. The distances, depths, and distance/depth transmission-loss values are stored in a `TL_Table` structure (Table 64 on page 141).

The second section of the TLM contains transmission-loss modeling-error data. The modeling errors are estimates of the transmission-loss values and may be distance/depth dependent, distance dependent only, or they may be condensed into a single, global value representing the modeling error for the entire TLM. `read_tl_table()` parses the distance and depth samples for which the modeling errors are estimated and then parses the modeling errors themselves. The distances, depths, and modeling errors are stored in a `TL_Mdl_Err` structure (Table 62 on page 140) nested within the `TL_Table` structure (Table 64 on page 141).

`read_tl_table()` returns the output variables listed in Table 32 to `read_tlsf()`. The `TL_Table` structure is stored as a component of the earth-model-data memory store (M1 in Figures 8, 9, and 10; Table 8 on page 38) by `read_tlsf()`.

TABLE 32: OUTPUT VARIABLES FROM READ_TL_TABLE()

Type	Variable Name	Use	Description
`int`	*icode*	R	status code
`TL_Table **`	*tl_table_ptr*	A	pointer to pointer to `TL_Table` structure

Interfaces

Only `read_tlsf()` calls `read_tl_table()` (see Table 18 on page 63 for hierarchy). `read_tl_table()` does not call any lower-level *libmagnitude* processing units.

Error States

`read_tl_table()` is a function designed not to be called directly by an external application. As a result, most of the input arguments in Table 31 on page 86 are not checked within `read_tl_table()` to ensure valid content. Exceptions are `phase` and `chan`, which are checked for valid content.

`read_tl_table()` writes an error message to stderr, terminates all further processing within itself, and returns the appropriate error or warning status code (Table 25 on page 76) to `read_tlsf()` if it encounters an error or warning condition. If `read_tl_table()` reads, parses, and stores the data from the TLM without encountering any error or warning conditions, it returns an OK status code to `read_tlsf()`.

build_mag_obj()

`build_mag_obj()` is the external interface and primary *libmagnitude* processing unit for constructing and storing input event and magnitude specification data associated with a single event in a memory store. These data are stored in `Magnitude` objects.

Input/Processing/Output

`build_mag_obj()` is a core element of the network-magnitude mode of operation. The external interface is called by an application operating in this mode to initiate the *Build Magnitude Data Store* process (2.2 in Figure 10 on page 43). `build_mag_obj()` is also the internal function that actually stores most of the event data and magnitude specification data in internal memory.

Table 33 describes the input variables to `build_mag_obj()`. The data source for these variables is the station, event, and control-parameter data memory store (M6 in Figure 10; Table 10 on page 44). Refer to Table 2 on page 14 and [IDC5.1.1Rev2] for descriptions of the database table structures and their corresponding schema, respectively.

TABLE 33: INPUT VARIABLES TO BUILD_MAG_OBJ()

Type	Variable Name	Use	Description
`char **`	*list_of_magtypes*	A	list of magtypes to be estimated
`int`	*num_magtypes*	A	number of elements in list of magtypes
`Origin *`	*origin*	A	pointer to `Origin` database table structure
`Netmag *`	*in_netmag*	A	array of `Netmag` database table structures
`int`	*num_netmags*	A	number of elements in array of `Netmag` structure

TABLE 33: INPUT VARIABLES TO BUILD_MAG_OBJ() (CONTINUED)

Type	Variable Name	Use	Description
`Stamag *`	*in_stamag*	A	array of `Stamag` database table structures
`int`	*num_stamags*	A	number of elements in array of `Stamag` structures
`Amplitude *`	*det_amplitude*	A	array of arrival-based `Amplitude` database table structures
`int`	*num_det_amps*	A	number of elements in array of arrival-based `Amplitude` structures
`Amplitude *`	*ev_amplitude*	A	array of origin-based `Amplitude` database table structures
`int`	*num_ev_amps*	A	number of elements in array of origin-based `Amplitude` structures
`Assoc *`	*in_assoc*	A	array of `Assoc` database table structures
`int`	*num_assocs*	A	number of elements in array of `Assoc` structures
`Parrival *`	*in_parrival*	A	array of `Parrival` database table structures
`int`	*num_parrivals*	A	number of elements in array of `Parrival` structures

`build_mag_obj()` stores input magnitude specification data and event data in an array of `Magnitude` objects (Table 54 on page 133) for a single event. Each element of the array of `Magnitude` objects is associated to one magtype passed in through the *list_of_magtypes* input variable. `Magnitude` objects provide a convenient way to bind amplitude and magnitude data together by magtype and to pass these data between applications and *libmagnitude* processing units.

The magnitude specification data are magnitude control settings for specific magtypes and are identical to some of the magnitude description data that `read_mdf()` stores in the array of `Mag_Descrip` structures (Table 55 on page 134; process 2.1 in Figure 10 on page 43). For each magtype listed in the `list_of_magtypes`, `build_mag_obj()` calls `get_magtypes_features()` (see Table 19 on page 65 for hierarchy) to copy the first 11 members of the appropriate `Mag_`

Descrip structure into a Mag_Cntrl structure nested within the associated element of the array of Magnitude objects. (Refer to the description of the Mag_Descrip structure (Table 55) for a description of the Mag_Cntrl structure.) The Mag_Cntrl structures contain magnitude specification data that an application may modify for each processed event and magtype.

build_mag_obj() also stores event data in the array of Magnitude objects. The event data consist of the input database table structures listed in Table 33 on page 88. For each input magtype, certain element addresses of input Amplitude and Stamag database table structures are stored in elements of the array of pointers to Amplitude and Stamag structures within the Magnitude object. The Amplitude addresses that are stored are those of the input Amplitude elements that have amptype members matching either the arrival-based or origin-based amptypes specified in the det_amptype or ev_amptype members of the Mag_Cntrl element. The stored Stamag addresses are those of the input Stamag elements whose magtypes are identical to the given magtype being processed. Similarly, the element of the Netmag structures, whose magtype is identical to the magtype currently being processed, is stored in the Netmag structure attached to the Magnitude object. The contents of such Stamag elements and Netmag members are termed "pre-existing."

In some cases, no input Stamag elements or Netmag component members are associated with a particular magtype. In such cases, build_mag_obj() populates the Stamag elements and Netmag component members of the Magnitude object with data from the input Origin, Assoc, Amplitude, and Parrival structures. Data from the Assoc structures are copied if the amptypes are arrival-based, and data from the Parrival structures are copied if the amptypes are origin-based. The auth member of each Stamag element is populated with the string "build_mag_obj" to indicate that these elements were created by build_mag_obj() during processing and were not retrieved from the database.

The event data stored in the Magnitude object for each magtype includes auxiliary station-magnitude data. build_mag_obj() uses data from each Amplitude element to populate corresponding elements in an array of auxiliary station-magnitude structures (SM_Aux; Table 57 on page 136).

An important `build_mag_obj()` and *libmagnitude* design decision is that the data model must possess at least one `Netmag` structure, which corresponds to any `Stamag` structures for a given magtype. That is, an array of pointers to `Stamag` structures, without an associated `Netmag` structure, violates the *libmagnitude* data model. Input data such as this cause *libmagnitude* to produce an incomplete `Netmag` structure member of the `Magnitude` object that contains `N/A` values in the `evid` and `magtype` fields. A `Netmag` structure without an associated array of pointers to `Stamag` structures also violates the data model. However, `build_mag_obj()` controls this situation by copying data from other database table structures into the `Stamag` elements of the `Magnitude` object.

`build_mag_obj()` stores the array of `Magnitude` objects in the magnitude-data memory store (M2 in Figure 10; Table 11 on page 45). The only output returned to the calling application from `build_mag_obj()` is a pointer to the array of `Magnitude` objects.

Interfaces

All external applications that operate in network-magnitude mode and need to store event and magnitude data in a memory store must call *libmagnitude* through `build_mag_obj()`, as indicated in Figure 10. *EvLoc* and *ARS* call `build_mag_obj()`. `build_mag_obj()` should be called once for each event processed. Applications that operate in station-magnitude mode do not require `build_mag_obj()`.

`build_mag_obj()` calls the lower-level *libmagnitude* processing units listed in Table 19 on page 65.

Error States

`build_mag_obj()` is an external interface and is designed to be called directly by an external application. As a result, each magtype contained in the `list_of_magtypes` input argument (Table 33 on page 88) is checked within the lower-level function `get_magtype_features()` to ensure that magnitude specification data exists for it. The application is responsible for ensuring that the remaining input arguments contain valid data.

`build_mag_obj()` interprets the status code or value returned from all lower-level processing units it calls (see Table 19 on page 65 for hierarchy). OK status codes or acceptable return values indicate to `build_mag_obj()` that processing was successful. Error codes or out-of-bounds return values indicate to `build_mag_obj()` that it should skip part of its functionality due to insufficient data.

`build_mag_obj()` writes an error message to stderr and terminates all further processing associated with an input magtype if an error status code is returned from `get_magtype_features()`. The error message is of the form:

```
Magtype: <MAGTYPE> is not specified within MDF
Hence, this magnitude cannot be computed!
```

where <*MAGTYPE*> is the magtype being processed.

`build_mag_obj()` checks for memory allocation errors for the array of `Magnitude` objects. If a memory allocation error occurs, `build_mag_obj()` returns a NULL pointer to the calling application.

station_magnitude()

`station_magnitude()` is the external or internal interface (depending on which mode the application is operating in) and primary *libmagnitude* processing unit for computing a station magnitude and uncertainty.

Input/Processing/Output

`station_magnitude()` is a core element of both the station- and network-magnitude modes of operation. The external interface is called by an application operating in station-magnitude mode to initiate the *Estimate Station-magnitude Data* process (2.3 in Figure 8 on page 37). `station_magnitude()` is a interface that is called by `calc_mags()` when an application is operating in network-magnitude mode as part of the *Estimate Station-magnitude Data* process (2.3 in Figure 10 on page 43). `station_magnitude()` is also the internal function that estimates station-magnitude data in both modes of operation.

Table 34 describes the input variables to `station_magnitude()`. The data source for these variables depends on what mode the application is operating in. When the application is operating in station-magnitude mode, the data source for all input arguments is the station, event, and control-parameter data memory store (M6 in Figure 8 on page 37). When the application is operating in network-magnitude mode, the data source for the `depth` argument is also the station, event, and control-parameter data memory store (M6 in Figure 10 on page 43; Table 10 on page 44). The data source for the remaining arguments is the magnitude-data memory store (M2 in Figure 10; Table 11 on page 45). Regardless of the mode of operation, the data source for the input variables read from memory is the earth-model-data memory store (M1 in Figures 8 and 10; Table 8).

TABLE 34: INPUT VARIABLES TO STATION_MAGNITUDE()

Type	Variable Name	Use	Description
`char *`	*magtype*	A	magnitude descriptor
`char *`	*sta*	A	station code
`char *`	*phase*	A	phase name
`char *`	*chan*	A	channel identifier
`Bool`	*extrapolate*	A	permit extrapolation of TLMs? 0 = no, 1 = yes
`char *`	*ts_region*[1]	A	magnitude test-site region descriptor
`double`	*distance*	A	event-to-station distance (deg)
`double`	*ev_depth*	A	origin depth (km)
`double`	*amp*	A	measured amplitude (nm)
`double`	*period*	A	measured period (s)
`double`	*duration*	A	duration of amplitude window (s)
`double`	*snr*	A	signal-to-noise ratio (not used)

TABLE 34: INPUT VARIABLES TO STATION_MAGNITUDE() (CONTINUED)

Type	Variable Name	Use	Description
`Mag_Descrip *`	*mag_descrip*	M	array of `Mag_Descrip` structures
`Mag_Sta_TLType *`	*mag_sta_tltype*	M	array of `Mag_Sta_TLType` structures

1. This variable is not applicable to the IDC.

`station_magnitude()` estimates a station magnitude, uncertainty, and ancillary station-magnitude data given a single amplitude and magtype. `station_magnitude()` estimates an initial (non-path-corrected) station magnitude as a function of the *amp* and *period* input variables. If both variables are out of bounds, then the *duration* variable is substituted to estimate the initial station magnitude.

`station_magnitude()` estimates the final station magnitude by applying two magnitude corrections to the initial station magnitude. The first magnitude correction is a distance/depth-dependent transmission-loss correction. `station_magnitude()` passes the *magtype*, *sta*, *phase*, *chan*, *distance*, *ev_depth*, and *extrapolate* variables to the lower-level functions `get_mag_indexes()` and `interp_for_tl_value()`. `interp_for_tl_value()` retrieves the distance/depth-dependent correction associated with the magtype-TLtype, station, phase name, and channel identifier inputs from the correct `TL_Table` structure (Table 64 on page 141) and returns it to `station_magnitude()`. The `TL_Table` structures are a component of the earth-model-data memory store (M1 in Figures 8 and 10; Table 8).

The second magnitude correction is a bulk station correction. `station_magnitude()` retrieves the bulk station correction associated with the input magtype-TLtype and the station from the input array of `Mag_Sta_TLType` structures (Table 56 on page 135). If an input magtype-TLtype and station combination does not have a bulk station correction associated with it, then `station_magnitude()` retrieves a default bulk station correction from the input array of `Mag_Descrip` structures (Table 55 on page 134).

`station_magnitude()` also estimates a station-magnitude uncertainty. The uncertainty is a root-mean-square (RMS) measure of a transmission-loss modeling error and the bulk-station-correction error. To obtain a transmission-loss modeling error, `station_magnitude()` passes the *magtype*, *sta*, *phase*, *chan*, *distance*, and *ev_depth* to the lower-level functions `get_mag_indexes()` and `get_tl_model_error()`. `get_tl_model_error()` retrieves the modeling error associated with the magtype-TLtype, station, phase name, and channel identifier inputs from the `TL_Mdl_Err` member (Table 62 on page 140) for the correct `TL_Table` structure (Table 64 on page 141) and returns it to `station_magnitude()`. `station_magnitude()` retrieves the bulk-station-correction error similar to retrieval of a bulk station correction.

`station_magnitude()` stores the station magnitude, uncertainty, magnitude corrections, transmission-loss modeling error, and bulk-station-correction error data in a `SM_Info` structure (Table 58 on page 136).

`station_magnitude()` returns the output variables listed in Table 35. If the calling application operates in station-magnitude mode, then `station_magnitude()` stores the `SM_Info` structure in the station-magnitude-data memory store (M7 in Figure 8 on page 37; Table 9 on page 40) and returns a `SM_Info` structure pointer to the application. If the application operates in network-magnitude mode, then the scope of the `SM_Info` structure is entirely within the *Estimate Station-magnitude Data* process (2.3 in Figure 10 on page 43), so the structure is not shown as a component of a memory store. `station_magnitude()` returns a `SM_Info` structure pointer to `calc_mags()` in this latter case.

TABLE 35: OUTPUT VARIABLES FROM STATION_MAGNITUDE()

Type	Variable Name	Use	Description
`double`	*sta_magnitude*	R	station magnitude
`SM_Info *`	*sm_info*	A	pointer to `SM_Info` structure

Interfaces

All external applications that operate in station-magnitude mode must call *libmagnitude* through `station_magnitude()` (Figure 8 on page 37). *StaPro*, the GA Subsystem (including *GAcons*), and *WaveExpert* all call `station_magnitude()`. `station_magnitude()` should be called once for each amplitude/magtype processed.

All applications that operate in network-magnitude mode do not directly call `station_magnitude()`. These applications call `calc_mags()` (Figure 10 on page 43), which subsequently calls `station_magnitude()`.

`station_magnitude()` calls the lower-level *libmagnitude* processing units listed in Table 20 on page 66.

Error States

`station_magnitude()` is an external interface when an external application operates in station-magnitude mode, and it is a function when an application operates in network-magnitude mode. Because `station_magnitude()` may be called directly by an application, several of its input arguments (Table 34 on page 93) are checked to ensure that they contain valid data. `station_magnitude()` checks the *amp* and *period* arguments. The *get_mag_indexes()* function checks the `magtype`, `sta`, `phase`, and `chan` arguments. The `interp_for_tl_value()` and `get_tl_model_error()` functions independently check the `distance` argument. The application is responsible for ensuring the remaining input arguments contain valid data.

`station_magnitude()` interprets the return value returned from all lower-level processing units (see Table 20 on page 66 for hierarchy). Acceptable return values indicate to `station_magnitude()` that processing was successful. Out-of-bounds return values from `get_mag_indexes()` or `interp_for_tl_value()` indicate to `station_magnitude()` that it should terminate all further processing. `station_magnitude()` returns an `N/A` station magnitude to the calling application or `calc_mags()`. An N/A value returned from `get_tl_model_error()` indicates that a modeling error could not be retrieved from the `TL_Table` structure. `station_magnitude()` changes the modeling error from an `N/A` value to the value of the `sgbase` member in the `Mag_Descrip` structure corresponding to the magtype being processed. *initialize_sm_info()* does not identify or return any possible error conditions. *get_meas_error()* is a dummy function that does not perform any processing.

calc_mags()

`calc_mags()` is the external network-magnitude processing interface between applications operating in network-magnitude mode and the *libmagnitude* processing units that estimate station- and network-magnitude data. `calc_mags()` also serves as a function that processes and stores station- and network-magnitude data.

Input/Processing/Output

`calc_mags()` is a core element of the network-magnitude mode of operation. The external interface is called by an application operating in this mode to initiate both the *Estimate Station-magnitude Data* process (2.3 in Figure 10 on page 43) and the *Estimate Network-magnitude Data* process (2.4 in Figure 10). `calc_mags()` is also the internal function that processes station- and network-magnitude data and stores these data in two memory stores.

Table 36 describes the input variables to `calc_mags()`. The data source for the pointer to the `Magnitude` object is the magnitude-data memory store (M2 in Figure 10; Table 11 on page 45). The data source for the three remaining arguments is the station, event, and control-parameter data memory store (M6 in Figure 10; Table 10 on page 44). Refer to Table 2 on page 14 and [IDC5.1.1Rev2] for descriptions of the `Origin` database table structure and its corresponding schema, respectively.

TABLE 36: INPUT VARIABLES TO CALC_MAGS()

Type	Variable Name	Use	Description
`Magnitude *`	*magn_ptr*	A	pointer to `Magnitude` object
`int`	*num_magns*	A	number of magtypes to be estimated
`Origin *`	*origin*	A	pointer to `Origin` database table structure
`Mag_Params *`	*mag_params*	A	pointer to `Mag_Params` structure

`calc_mags()` processes station- and network-magnitude data for each magtype given an event. This processing includes calling `station_magnitude()` to estimate station-magnitude data, identifying the magnitude-defining state of the resulting station-magnitude data, estimating network magnitudes and uncertainties from the magnitude-defining data, and storing the resulting station- and network-magnitude data in two memory stores. The settings of certain magnitude control parameters stored in the `Mag_Params` input structure (Table 51 on page 129) control how `calc_mags()` processes the station- and network-magnitude data.

`calc_mags()` initiates the *Estimate Station-magnitude Data* process by calling `station_magnitude()` to estimate station magnitudes and uncertainties for all arrival-based and origin-based amplitudes associated with the current magtype being processed. `station_magnitude()` returns each station magnitude to `calc_mags()` as the return value (Table 35 on page 95); each uncertainty is returned through the `SM_Info` structure (Table 35 and Table 58 on page 136). `calc_mags()` copies the station magnitudes into the `magnitude` members of the `Stamag` database table structures (Table 2 on page 14; [IDC5.1.1Rev2]) nested in the `Magnitude` objects. The uncertainties are copied from the `model_plus_meas_error` members of the `SM_Info` structures into the `uncertainty` members of the `Stamag` structures if they are positive values and if weighted network-average or weighted-MLE magnitudes will be requested. Otherwise, the `N/A` uncertainty is stored in the `Stamag` structures.

calc_mags() identifies the magnitude-defining state of each station magnitude and uncertainty for a given magtype using several criteria. A "magnitude-defining" state means that the station magnitude and uncertainty will be used to estimate a network magnitude and uncertainty. A "magnitude-nondefining" state means that the station magnitude and uncertainty will not be used to determine the network magnitude and uncertainty. All station magnitudes and uncertainties are assumed to be magnitude-defining before the state-defining criteria are applied. Applications may override the criteria for a particular amplitude or station magnitude by switching the manual_override flag in the SM_Aux member (Table 57 on page 136) of the Magnitude object to TRUE after calling build_mag_obj(), but prior to calling calc_mags(). This design feature is not indicated in Figure 10 on page 43. The states of the station magnitudes are recorded in the magdef members of the Stamag structures. calc_mags() also updates the delta and mmodel members of the Stamag structures. Finally, calc_mags() copies each station magnitude, uncertainty, magnitude-defining state, and signal type into an array of SM_Sub structures (Table 59 on page 138), which carries the station-magnitude data amongst functions in the *Estimate Network-magnitude Data* process.

All processing of the station-magnitude data are now complete. At this point, the *Estimate Station-magnitude Data* process ends and the *Estimate Network-magnitude Data* process begins (Figure 10 on page 43 and Figure 11 on page 48). calc_mags() calls other *libmagnitude* processing units (see Table 21 on page 68 for hierarchy) to estimate network magnitudes and uncertainties using the magnitude-defining station magnitudes and uncertainties associated with each magtype. *libmagnitude* functions may estimate weighted or unweighted network-average, MLE, and upper- or lower-bound magnitudes and uncertainties ("Chapter 2: Architectural Design" on page 9). calc_mags() calls network_mag() to estimate initial network-average, MLE, and upper- or lower-bound magnitudes and uncertainties. calc_mags() optionally calls mag_boot_strap() to estimate MLE magnitudes, standard deviations, and uncertainties via bootstrap resampling [McL88] of the defining station magnitudes and uncertainties.

After an initial network magnitude is estimated, `calc_mags()` computes the station-magnitude residuals (Figure 11 on page 48). `calc_mags()` stores the residuals in the `magres` members of the `Stamag` structures. If any outlying residuals are found, they may be optionally screened all at once and new network-magnitude data may be estimated. The outlying station-magnitude data are screened from further network magnitude calculations by changing their state to magnitude-non-defining, and the `magdef` members of `Stamag` are set to `n`. This optional screening process repeats until all outliers have been removed. The result is a final network magnitude and uncertainty. Former outliers cannot be restored. Station magnitudes whose associated `manual_override` flags (in the `SM_Aux` structures) are set to TRUE are exempt from this outlier screening process.

In general, `calc_mags()` copies the final network magnitudes and uncertainties into the `magnitude` and `uncertainty` members of the `Netmag` structures (Table 2 on page 14; [IDC5.1.1Rev2]) nested within the `Magnitude` objects. The final network magnitudes and uncertainties are identical to the initial network magnitudes and uncertainties if outlier screening was not performed. If MLE-magnitude data were estimated via bootstrap resampling, then only the bootstrapped uncertainties are stored in the `uncertainty` members of the `Netmag` structure. The remaining members of `Netmag` are populated with MLE-magnitude data estimated without bootstrap resampling. The final network magnitudes and uncertainties are not stored in `Netmag` when no magnitude-defining station magnitudes remain after the outlier screening process. In this case, `N/A` values for the final magnitude and uncertainty are stored in `Netmag`. `calc_mags()` also updates the `net`, `orid`, and `nsta` members of `Netmag`.

`calc_mags()` stores the station- and network-magnitude data in two memory stores. As mentioned, the station- and network-magnitude data are stored in the `Stamag` and `Netmag` members of the `Magnitude` objects within the magnitude-data memory store (M2 in Figure 10 and Figure 11; Table 11 on page 45). In addition, if the calling application optionally wants to retain the network magnitudes in an output **origin** database table, then the network magnitudes themselves are also stored in `Origin` structures within the station, event, and control-parameter data memory store (M6 in Figure 10 and Figure 11; Table 10 on page 44). `calc_mags()` may update the `mb`, `mbid`, `ms`, `msid`, `ml`, and `mlid` members of the `Origin` structure (Table 49 on page 121).

`calc_mags()` returns the output variables listed in Table 37 to the calling application.

TABLE 37: OUTPUT VARIABLES FROM CALC_MAGS()

Type	Variable Name	Use	Description
`int`	`num_mags`	R	number of network magnitudes stored in `Netmag` database table structure member of `Magnitude` object
`Magnitude *`	`magn_ptr`	A	pointer to updated `Magnitude` object
`Origin *`	`origin`	A	pointer to updated `Origin` database table structure

Interfaces

All external applications that operate in network-magnitude mode must call *libmagnitude* through `calc_mags()`, as indicated in Figure 10. *EvLoc* and *ARS* call `calc_mags()`. `calc_mags()` should be called once for each origin processed, although many magtypes may be processed per event. Applications that operate in station-magnitude mode do not need to call `calc_mags()`.

`calc_mags()` calls the lower-level *libmagnitude* processing units listed in Table 21 on page 68.

Error States

`calc_mags()` is an external interface and is designed to be called directly by an external application. The application is responsible for ensuring that the input arguments (Table 36 on page 98) contain valid data.

network_mag(), mag_boot_strap(), mag_max_lik(), and only_bound_amps() return an integer status code to a higher-level interface (see Table 21 on page 68 for hierarchy). The higher-level interfaces interpret the status code and determine whether or not to continue processing. Table 38 describes these status codes, which are associated with computing a network magnitude and uncertainty. The Status Type column is either "OK" or "Warning." The software within the *Estimate Network-magnitude Data* process handles potentially erroneous conditions without needing to terminate the process. An OK status type indicates to the higher-level interface that it should continue its processing because processing in the lower-level function was successful. A warning status type indicates to the higher-level interface that the network magnitude estimated in the lower-level interface was not reliable. The Status Message Description column in Table 21 describes the message associated with the status code. These strings are not written to stderr, but a similar message may be written by the processing unit that encounters the warning condition. The Source Processing Unit(s) column lists the processing unit(s) that may encounter a condition producing the status code.

TABLE 38: ESTIMATE NETWORK-MAGNITUDE DATA PROCESS STATUS CODES

Status Code	Status Type	Status Message Description	Source Processing Unit(s)
0	OK	successful magnitude and uncertainty calculations completed	network_mag(), mag_boot_strap(), mag_max_lik(), only_bound_amps()
–1	Warning	no station-magnitude data available	network_mag()
–2	Warning	maximum number of allowable iterations exceeded	mag_max_lik(), only_bound_amps()
–3	Warning	maximum number of allowable iterations exceeded while trying to estimate upper-bound magnitude and uncertainty	network_mag()
–4	Warning	maximum number of allowable iterations exceeded while trying to estimate lower-bound magnitude and uncertainty	network_mag()

TABLE 38: ESTIMATE NETWORK-MAGNITUDE DATA PROCESS STATUS CODES

Status Code	Status Type	Status Message Description	Source Processing Unit(s)
1	OK	only origin-based amplitudes available	`network_mag()`
2	OK	only clipped amplitudes available	`network_mag()`

`calc_mags()` interprets the `station_magnitude()` return value (see Table 21 on page 68). Acceptable return values indicate to `calc_mags()` that processing was successful. An `N/A` value returned from `station_magnitude()` indicates to `calc_mags()` that a station magnitude could not be estimated. `calc_mags()` makes that station magnitude nondefining and continues its processing.

`calc_mags()` also optionally writes station- and network-magnitude data to stdout.

network_mag()

`network_mag()` is a function that estimates a network-average magnitude and uncertainty for a single magtype from defining station-magnitude data. `network_mag()` also calls other *libmagnitude* functions to estimate MLE and upper- or lower-bound magnitudes and standard deviations.

Input/Processing/Output

`network_mag()` is a function that is called by `calc_mags()` as part of the *Estimate Network-magnitude Data* process. See Figure 10 on page 43 for the relationship of this process to applications and other *libmagnitude* processes. See Figure 11 on page 48 and Table 21 on page 68 for the relationship between this function and other *libmagnitude* functions within the *Estimate Network-magnitude Data* process.

Table 39 describes the input variables to `network_mag()`. The data source for the *verbose* variable is the station, event, and control-parameter data memory store (M6 in Figure 11; Table 10 on page 44). The data source for the three remaining variables is the magnitude-data memory store (M2 in Figure 11; Table 11 on page 45).

TABLE 39: INPUT VARIABLES TO NETWORK_MAG()

Type	Variable Name	Use	Description
`SM_Sub *`	*sm_sub*	A	array of `SM_Sub` structures
`Mag_Cntrl *`	*mcntrl*	A	pointer to `Mag_Cntrl` structure
`int`	*sm_count*	A	number of elements in array of `SM_Sub` structures
`int`	*verbose*	A	level of verbosity for printed magnitude output

`network_mag()` estimates a weighted or unweighted network-average magnitude and uncertainty. The inputs to the network-average magnitude estimates are a magtype, magnitude-defining station magnitudes, and uncertainties computed from arrival-based amplitudes. `network_mag()` also calls other *libmagnitude* functions to estimate a weighted or unweighted MLE, upper-bound magnitude, lower-bound magnitude, and associated standard deviations. The input array of `SM_Sub` structures (Table 59 on page 138) contains the station magnitudes, uncertainties, magnitude-defining states, and signal types associated with the magtype and event. Weighted network-average magnitude data are determined using the station-magnitude uncertainties as weights.

`network_mag()` estimates a single weighted or unweighted network-average magnitude using only magnitude-defining station-magnitude data estimated from arrival-based amplitudes. `network_mag()` constrains the uncertainty (standard deviation) of the network average to be within the lower- and upper-standard deviation bounds defined in the MDF, if these two bounds are not identical. These bounds are contained in the `sglim1` and `sglim2` members of the `Mag_Cntrl` structure, respectively (see Table 55 on page 134). If the standard deviation is out of the allowed range, then it is reset to the value of the nearest boundary. The standard deviation of the network average is not constrained if the lower- and upper-standard deviation bounds are identical.

If only one magnitude-defining station magnitude is available, then `network_mag()` equates the network-average magnitude with the station magnitude and defines the standard deviation of the network average to be the standard deviation baseline value for the magtype being processed. The baseline value is defined in the MDF and stored in the `sgbase` member of the `Mag_Cntrl` structure.

`network_mag()` also calls `mag_max_lik()` and `only_bound_amps()` to estimate a single weighted or unweighted MLE and an upper- or lower-bound magnitude and standard deviation, respectively. Only magnitude-defining station magnitudes estimated from a combination of arrival-based, origin-based, and clipped amplitudes are used to estimate a MLE magnitude and standard deviation. Only magnitude-defining station magnitudes estimated from origin-based amplitudes are used to estimate an upper-bound magnitude and standard deviation. Only magnitude-defining station magnitudes estimated from clipped amplitudes are used to estimate a lower-bound magnitude and standard deviation. Clipped amplitude data are seldom encountered in routine IDC processing.

`network_mag()` estimates an MLE and an upper- or lower-bound trial magnitude and standard deviation using a set of magnitude-defining station-magnitude data. These trial values are passed to `mag_max_lik()` and `only_bound_amps()`. The magnitude and standard deviation estimates computed by `mag_max_lik()` and `only_bound_amps()` are returned to `network_mag()`.

`network_mag()` returns the output variables listed in Table 40 to `calc_mags()`. The *mag*, *sigma*, and *sdav* arguments contain the magnitude and uncertainty estimates associated with the magnitude type that was computed.

TABLE 40: OUTPUT VARIABLES FROM NETWORK_MAG()

Name	Type	Use	Description
icode	`int`	R	status code
mag	`double *`	A	network magnitude

TABLE 40: OUTPUT VARIABLES FROM NETWORK_MAG() (CONTINUED)

Name	Type	Use	Description
sigma	`double *`	A	network-magnitude standard deviation
sdav	`double *`	A	network-magnitude uncertainty
num_amps_used	`int *`	A	number of magnitude-defining station magnitudes used to estimate network-magnitude data

Interfaces

Only `calc_mags()` calls `network_mag()` (see Table 21 on page 68 for hierarchy). `network_mag()` calls the lower-level *libmagnitude* processing units `mag_max_lik()` and `only_bound_amps()`.

Error States

`network_mag()` is a function designed not to be called directly by an external application. As a result, the input arguments in Table 39 are not checked within `network_mag()` to ensure valid content.

`network_mag()` interprets the status code returned from `only_bound_amps()` (see Table 21 for hierarchy and Table 38 on page 102 for the status code descriptions). An OK status code indicates to `network_mag()` that processing was successful. A warning status code indicates to `network_mag()` that it should terminate all further processing and return a warning code to `calc_mags()`. `network_mag()` returns a –3 or –4 warning code depending on whether an upper- or lower-magnitude bound is estimated.

`network_mag()` does not interpret the status code returned from `mag_max_lik()`. `network_mag()` continues its processing regardless of the status code returned by `mag_max_lik()`. This is acceptable even if `mag_max_lik()` encountered a warning condition and returned a warning status code to `network_mag()`, because the magnitude and standard deviation returned from `mag_max_lik()` are the best MLE estimates available.

`network_mag()` ensures that the network-average standard deviation is constrained within `sglim1` and `sglim2` as long as these two bounds are not identical. If the standard deviation of the network average is outside of the allowable range, then `network_mag()` resets the standard deviation to be the value of the nearest bound (either `sglim1` or `sglim2`), writes a warning message to stdout, and continues its processing. The warning message is of the form:

`Warning: Network stdev =` *STDEV* `< lower bound in mdf file =` *SGLIM1* `->`

`Setting network sigma =` *SGLIM1*

or

`Warning: Network stdev =` *STDEV* `> upper bound in mdf file =` *SGLIM2* `->`

`Setting network sigma =` *SGLIM2*

depending on whether or not the standard deviation is less than *sglim1* or greater than *sglim2*. *STDEV* is the standard deviation of the network average, and *SGLIM1* and *SGLIM2* are the initial lower- and upper-bound standard deviations, respectively.

mag_boot_strap()

`mag_boot_strap()` is a function that uses the bootstrap method to estimate an MLE magnitude, standard deviation, and uncertainty for a single magtype from magnitude-defining station-magnitude data.

Input/Processing/Output

`mag_boot_strap()` is a function that is called by `calc_mags()` as part of the *Estimate Network-magnitude Data* process. See Figure 10 on page 43 for the relationship of this process to applications and other *libmagnitude* processes. See Figure 11 on page 48 and Table 21 on page 68 for the relationship between this function and other *libmagnitude* functions within the *Estimate Network-magnitude Data* process.

Table 41 describes the input variables to `mag_boot_strap()`. The data source for the *num_boots* and *verbose* variables is the station, event, and control-parameter data memory store (M6 in Figure 11; Table 10 on page 44). The data source for the *sm_sub*, *mcntrl*, and *sm_count* variables is the magnitude-data memory store (M2 in Figure 11; Table 11 on page 45). `calc_mags()` stores values in the remaining variables prior to calling `mag_boot_strap()`.

TABLE 41: INPUT VARIABLES TO MAG_BOOT_STRAP()

Type	Name	Use	Description
`SM_Sub *`	*sm_sub*	A	array of `SM_Sub` structures
`Mag_Cntrl *`	*mcntrl*	A	pointer to `Mag_Cntrl` structure
`int`	*sm_count*	A	number of elements in array of `SM_Sub` structures
`int`	*num_boots*	A	maximum number of bootstrap resamples permitted
`double`	*net_mag*	A	trial network-average magnitude
`double`	*sigma*	A	trial network-average standard deviation
`int`	*verbose*	A	level of verbosity for printed magnitude output

`mag_boot_strap()` uses the bootstrap method [McL88] to estimate a weighted or unweighted MLE magnitude, standard deviation, and two uncertainties for a single magtype. This bootstrap procedure begins by randomly resampling (with repetition) the input magnitude-defining station-magnitude data stored in the input array of `SM_Sub` structures (Table 59 on page 138). At least one of the resampled defining station-magnitude data elements must have been estimated from an arrival-based amplitude, but the remainder may be any combination of station-magnitude data estimated from arrival-based, origin-based, and clipped amplitudes. Weighted MLE-magnitude data are determined using the station-magnitude uncertainties as weights.

`mag_boot_strap()` passes the resampled station-magnitude data to `mag_max_lik()`, along with the trial magnitude and standard deviation stored in the input variables *net_mag* and *sigma* and the pointer to the `Mag_Cntrl` structure. `mag_max_lik()` estimates and returns an MLE magnitude and standard deviation for this station-magnitude data set.

`mag_boot_strap()` repeats the above steps at least 10 times. After each MLE magnitude and standard deviation is returned from `mag_max_lik()`, `mag_boot_strap()` adds them to distributions of previously determined magnitudes and standard deviations and computes an average and standard deviation of each distribution. The uncertainties in the magnitude and standard deviation are defined to be the standard deviations of each distribution. This resampling, distributing, and estimating process continues until the convergence criteria are met or until the maximum number of bootstrap resamples (stored in the *num_boots* input variable) is reached.

`mag_boot_strap()` returns the output variables listed in Table 42 to `calc_mags()`. The *fmag1* and *sig1* arguments contain the averages of the magnitude and standard deviation distributions, respectively. The *sigmu* and *sigsig* arguments contain the uncertainties in the magnitude and standard deviation distributions, respectively.

TABLE 42: OUTPUT VARIABLES FROM MAG_BOOT_STRAP()

Type	Name	Use	Description
`int`	*icode*	R	status code
`double *`	*fmag1*	A	bootstrapped MLE magnitude
`double *`	*sigmu*	A	uncertainty in bootstrapped MLE magnitude
`double *`	*sig1*	A	bootstrapped MLE standard deviation
`double *`	*sigsig*	A	uncertainty in bootstrapped MLE standard deviation

Interfaces

Only calc_mags() calls mag_boot_strap() (see Table 21 on page 68 for hierarchy). mag_boot_strap() only calls the lower-level *libmagnitude* processing unit mag_max_lik().

Error States

mag_boot_strap() is a function designed not to be called directly by an external application. As a result, the input arguments in Table 41 are not checked within mag_boot_strap() to ensure valid content.

mag_boot_strap() does not interpret the status code returned from mag_max_lik(). mag_boot_strap() continues its processing regardless of the status code returned by mag_max_lik(). This is acceptable even if mag_max_lik() encountered a warning condition and returned a warning status code to network_mag() because the magnitude and standard deviation returned from mag_max_lik() are the best MLE estimates available.

mag_boot_strap() optionally writes bootstrapped MLE network-magnitude data to stdout.

mag_max_lik()

mag_max_lik() is a function that estimates an MLE magnitude and standard deviation for a single magtype given magnitude-defining station-magnitude data.

Input/Processing/Output

mag_max_lik() is a function that is called by network_mag() and mag_boot_strap() as part of the *Estimate Network-magnitude Data* process. See Figure 10 on page 43 for the relationship of this process to applications and other *libmagnitude* processes. See Figure 11 on page 48 and Table 21 on page 68 for the relationship between this function and other *libmagnitude* functions within the *Estimate Network-magnitude Data* process.

Table 43 describes the input variables to `mag_max_lik()`. The data source for the *verbose* variable is the station, event, and control-parameter data memory store (M6 in Figure 11; Table 10 on page 44). The data source for the *sm_sub*, *mcntrl*, and *sm_count* variables is the magnitude-data memory store (M2 in Figure 11; Table 11 on page 45). The calling interfaces store values in the remaining variables prior to calling `mag_max_lik()`.

TABLE 43: INPUT VARIABLES TO MAG_MAX_LIK()

Type	Name	Use	Description
`SM_Sub *`	*sm_sub*	A	array of `SM_Sub` structures
`Mag_Cntrl *`	*mcntrl*	A	pointer to `Mag_Cntrl` structure
`int`	*sm_count*	A	number of elements in array of `SM_Sub` structures
`double`	*ave*	A	trial network-average magnitude
`double *`	*net_mag*	A	trial network-average magnitude
`double *`	*sigma*	A	trial network-average standard deviation
`int`	*verbose*	A	level of verbosity for printed magnitude output

`mag_max_lik()` uses an iterative Expectation-Maximization (EM) algorithm [Bla82] to estimate a weighted or unweighted MLE magnitude and standard deviation for each specified magtype from magnitude-defining station magnitudes and uncertainties. At least one of the defining station magnitudes must have been estimated from an arrival-based amplitude, but the remainder may be any combination of station-magnitude data estimated from arrival-based, origin-based, and clipped amplitudes. The station-magnitude data are retrieved from the input array of `SM_Sub` structures (Table 59 on page 138). Weighted MLE-magnitude data are determined using the station-magnitude uncertainties as weights.

The first iteration of the EM algorithm uses trial values for the initial MLE magnitude and standard deviation. The input variables *ave* and *net_mag* contain the initial trial network-average magnitudes. If *ave* and *net_mag* differ by more than one magnitude unit, then *ave* is used as the initial trial magnitude. Otherwise, *net_mag* is used. The input variable *sigma* contains the initial trial standard deviation.

At the completion of each iteration, `mag_max_lik()` constrains the MLE standard deviation of the MLE to be within the lower- and upper-standard deviation bounds defined for the magtype. These bounds are contained in the `sglim1` and `sglim2` members of the `Mag_Cntrl` structure, respectively (see Table 55 on page 134). If the standard deviation is out of the allowed range, then it is reset to the value of the nearest boundary. The iterations of the EM algorithm continue until convergence criteria are met or until the number of iterations exceeds 200.

If only one magnitude-defining station magnitude is available, then `mag_max_lik()` equates the MLE magnitude with the station magnitude and defines the MLE standard deviation to be the standard deviation baseline value for the magtype being processed. The baseline value is stored in the `sgbase` member of the `Mag_Cntrl` structure.

MLE-magnitude data may be estimated if the station-magnitude data are all estimated from arrival-based amplitudes. In this case, the MLE magnitude and standard deviation estimated by `mag_max_lik()` are identical to those estimated for the network average.

One important characteristic of an MLE is that it does not generally increase the precision of the network-magnitude estimate and may actually increase the uncertainty due to the introduction of origin-based and clipped amplitudes. However, the MLE does reduce systematic biases due to the origin-based and clipped measurements [McL88].

`mag_max_lik()` returns the output variables listed in Table 44 to the calling interface. *net_mag* and *sigma* are also input variables, but `mag_max_lik()` overwrites their input values with the MLE magnitude and standard deviation estimates.

TABLE 44: OUTPUT VARIABLES FROM MAG_MAX_LIK()

Type	Variable Name	Use	Description
`int`	*icode*	R	status code
`double *`	*net_mag*	A	MLE magnitude
`double *`	*sigma*	A	MLE standard deviation

Interfaces

Only `network_mag()` and `mag_boot_strap()` call `mag_max_lik()` (see Table 21 on page 68 for hierarchy). `mag_max_lik()` does not call any lower-level *libmagnitude* processing units.

Error States

`mag_max_lik()` is a function designed not to be called directly by an external application. As a result, the input arguments are not checked within `mag_max_lik()` to ensure valid content.

`mag_max_lik()` checks that the number of iterations in the EM algorithm is less than 200. If more than 200 iterations occur before the convergence criteria is satisfied, then `mag_max_lik()` writes a warning message to stderr and returns a warning status code of `-2` along with the last estimate of the MLE magnitude and standard deviation to the calling interface (see Table 38 on page 102 for status code descriptions). The warning message is:

```
EM ESTIMATOR HAS NOT CONVERGED AFTER 200 ITERATIONS!
```

only_bound_amps()

`only_bound_amps()` is a function that estimates an upper- or lower-bound magnitude and standard deviation for a single magtype from magnitude-defining station-magnitude data.

Input/Processing/Output

`only_bound_amps()` is a function that is called by `network_mag()` as part of the *Estimate Network-magnitude Data* process. See Figure 10 on page 43 for the relationship of this process to applications and other *libmagnitude* processes. See Figure 11 on page 48 and Table 21 on page 68 for the relationship between this function and other *libmagnitude* functions within the *Estimate Network-magnitude Data* process.

Table 45 describes the input variables to `only_bound_amps()`. The data source for the *sm_sub*, *mcntrl*, and *sub_count* variables is the magnitude-data memory store (M3 in Figure 11; Table 11 on page 45). `network_mag()` stores values in the remaining variables prior to calling `only_bound_amps()`.

TABLE 45: INPUT VARIABLES TO ONLY_BOUND_AMPS()

Type	Name	Use	Description
`SM_Sub *`	*sm_sub*	A	array of `SM_Sub` structures
`Mag_Cntrl *`	*mcntrl*	A	pointer to `Mag_Cntrl` structure
`int`	*sub_count*	A	number of elements in array of `SM_Sub` structures
`double`	*ave*	A	trial network-average magnitude
`int`	*isign*	A	indicates whether to estimate upper- or lower-magnitude bounds: –1 = upper bound, 1 = lower bound
`double`	*sigma*	A	trial network-average standard deviation

`only_bound_amps()` uses an interactive hypothesis test algorithm to estimate a weighted or unweighted upper- or lower-bound magnitude and standard deviation for a given magtype from magnitude-defining station magnitudes and uncertainties. The magnitude-defining station-magnitude data must be origin-based if an upper-bound magnitude is being estimated. The station-magnitude data must be clipped amplitudes if a lower-bound magnitude is being estimated. The station-magnitude data are retrieved from the input array of `SM_Sub` structures (Table 59 on page 138). Weighted upper- and lower-bound magnitude data are determined using the station-magnitude uncertainties as weights.

The hypothesis test algorithm initializes a standard deviation and performs a grid search over a set of trial magnitudes, beginning with a trial magnitude just below or above the trial value stored in the *ave* variable (depending on the value of the *isign* input variable). The standard deviation is set to the value of the `sglim2` member in the `Mag_Cntrl` structure. An initial trial standard deviation, stored in the *sigma* input variable, is presently not used in the algorithm.

For each trial magnitude, `only_bound_amps()` computes the probability that all input station magnitudes could not be generated from an event with that particular trial magnitude and standard deviation. The magnitude that rejects this hypothesis at the 95 percent confidence level is chosen as the upper- or lower-bound magnitude estimate.

`only_bound_amps()` returns the output variables listed in Table 46 to `network_mag()`. The *net_mag* and *sigmax* arguments contain the upper- or lower-bound magnitude estimate and fixed standard deviation, respectively.

TABLE 46: OUTPUT VARIABLES FROM ONLY_BOUND_AMPS()

Type	Name	Use	Description
`int`	*icode*	R	status code
`double *`	*net_mag*	A	upper- or lower-bound magnitude
`double *`	*sigmax*	A	upper- or lower-bound standard deviation

Interfaces

Only `network_mag()` calls `only_bound_amps()` (see Table 21 on page 68 for hierarchy). `only_bound_amps()` does not call any other lower-level *libmagnitude* processing units.

Error States

`only_bound_amps()` is a function designed not to be called directly by an external application. As a result, the input arguments are not checked within `only_bound_amps()` to ensure valid content.

`only_bound_amps()` checks that the number of hypotheses tested is less than 200. If 200 or more iterations are needed to reject the test hypothesis, then `only_bound_amps()` returns a warning status code of –2 along with the last estimate of the upper- or lower-bound magnitude and standard deviation to the calling interface (see Table 38 for status code descriptions).

PRIMARY LIBMAGNITUDE FUNCTIONAL AREAS

The *libmagnitude* processing units address two broad functional areas: (1) Station Magnitude Estimation and (2) Network Magnitude Estimation. The 23 Station Magnitude Estimation processing units are defined in 6 *libmagnitude* files. The 27 Network Magnitude Estimation processing units are defined 10 *libmagnitude* files. Two of the processing units, `get_delta_for_sta()` and `get_TL_indexes()`, address both functional areas. Two of the files, `TL_manipulation.c` and `mag_access.c`, define processing units of both functional areas. Altogether, 48 distinct *libmagnitude* processing units are defined in 14 distinct source-code files.

Station Magnitude Estimation

Table 47 lists the processing units and source-code files associated with the Station Magnitude Estimation functional area. The File Description column summarizes the functional scope of the source-code file. Some of the files contain only a single processing unit. Other files contain multiple processing units that are grouped together in the file based on commonly shared memory store components. For example, the eight processing units in `TL_manipulation.c` associated with estimating station-magnitude data all share components from the internal earth-model-data memory store (M1 in Figure 8 on page 37; Table 8 on page 38) that should not be under the control of external applications. All memory-store components associated with the processing units in Table 47 are stored in M1 or the external station-magnitude-data memory store (M7 in Figure 8; Table 9 on page 40). This table includes only the processing units that are accessed in both station- and network-magnitude modes, that is, those that compose the *Read Earth-model Data* process and *Estimate Station-magnitude Data* process (2.1 and 2.3 in Figure 10 on page 43).

TABLE 47: STATION MAGNITUDE ESTIMATION SOURCE-CODE FILES

Source-code File	Processing Units	File Description
`mag_access.c`	`setup_mag_facilities()`, `setup_mc_tables()`, `station_magnitude()`, `initialize_sm_info()`, `get_mag_indexes()`, `get_meas_error()`, `abbrev_sta_mag()`, `reset_max_dist()`, `reset_min_dist()`, `revert_max_dist()`, `revert_min_dist()`	provides core station-magnitude data handling facilities; `setup_mag_facilities()` and `setup_mc_tables()` call other processing units to read and store data from MDF, TLSF, and TLMs; the remaining processing units handle computation of station-magnitude data
`mag_error_msg.c`	`mag_error_msg()`	links magnitude status code with status message string
`read_mdf.c`	`read_mdf()`	reads and stores data from single MDF in data structures defined in include file `mag_descrip.h`

TABLE 47: STATION MAGNITUDE ESTIMATION SOURCE-CODE FILES (CONTINUED)

Source-code File	Processing Units	File Description
`TL_manipulation.c`	`set_sta_TL_pt()`, `read_tlsf()`, `free_tl_table()`, `get_delta_for_sta()`, `interp_for_tl_value()`, `get_TL_ts_corr()`, `get_tl_model_error()`, `get_TL_indexes()`	provides core transmission-loss data handling facilities. `read_tlsf()` reads and stores data from single TLSF in data structures defined in include file `tl_table.h`; the remaining processing units handle retrieval of transmission-loss data (magnitude correction data) from M1 in Figure 8 and Figure 10
`TL_error_msg.c`	`TL_error_msg()`	lInks transmission-loss status code with status message string
`read_tl_table.c`	`read_tl_table()`	reads and stores data from single TLM in data structures defined in include file `tl_table.h`

Network Magnitude Estimation

Table 48 lists the processing units and source-code files associated with the Network Magnitude Estimation functional area. As with the Station Magnitude Estimation functional area, some of the files listed in Table 48 contain only a single processing unit. Other files contain multiple processing units that are grouped together in the file based on commonly shared memory-store components. All components associated with the processing units in Table 48 are stored in the earth-model-data memory store or the external magnitude-data memory store (M2; Table 11 on page 45). This table includes only the processing units that are accessed in network-magnitude mode but not station-magnitude mode, that is, those that compose the *Build Magnitude Data Store* process and *Estimate Network-magnitude Data* process (2.2 and 2.4, respectively, in Figure 10 on page 43).

TABLE 48: NETWORK MAGNITUDE ESTIMATION SOURCE-CODE FILES

Source Code File	Processing Units	File Description
`build_mag_obj.c`	`build_mag_obj()`	stores event and magnitude specification data in `Magnitude` objects (Table 54 on page 133); this data structure is defined in include file `mag_descrip.h`
`mag_utils.c`	`copy_magnitudes()`, `free_magnitudes()`	provides processing units for copying and freeing memory allocated to `Magnitude` objects (Table 54)
`mag_access.c`	`get_magtype_features()`, `reset_algorithm()`, `reset_amptypes()`, `reset_sd_baseline()`, `reset_sd_limits()`, `reset_wgt_ave_flag()`, `revert_algorithm()`, `revert_amptypes()`, `revert_sd_baseline()`, `revert_sd_limits()`, `revert_wgt_ave_flag()`	provides processing units for initializing and modifying members of `Mag_Descrip` and `Mag_Cntrl` structures (Table 55 on page 134); this data structure is defined in include file `mag_descrip.h`
`TL_manipulation.c`	`valid_phase_for_TLtype()`, `valid_range_for_TLtable()`, `get_delta_for_sta()`, `get_TLMD_index()`, `get_TL_indexes()`	provides processing units for identifying which event data to store in `Magnitude` objects (Table 54)
`mag_params.c`	`initialize_mag_params()`	initializes `Mag_Params` structure (Table 51 on page 129) to default values; this data structure is defined in include file `mag_params.h`

TABLE 48: NETWORK MAGNITUDE ESTIMATION SOURCE-CODE FILES (CONTINUED)

Source Code File	Processing Units	File Description
calc_mags.c	calc_mags(),[1] mag_set_compute_upper_bounds(), mag_get_compute_upper_bounds()	provides several network-magnitude interfaces and processing units; calc_mags() is the primary interface and function for estimating and storing station- and network-magnitude data in network-magnitude mode; the other processing units determine whether upper-bound magnitudes should be estimated and stored
network_mag.c	network_mag()	estimates single network-average magnitude, standard deviation, and uncertainty
mag_boot_strap.c	mag_boot_strap()	estimates single MLE magnitude, standard deviation, and two uncertainties using bootstrap resampling
mag_max_lik.c	mag_max_lik()	estimates single MLE magnitude and standard deviation
only_bound_amps.c	only_bound_amps()	estimates single upper- or lower-bound magnitude and standard deviation

1. A small portion of this processing unit is used to determine station-magnitude data.

DATA DESCRIPTION

EvLoc reads station and event data from an input database account and writes magnitude results to an output database account. It interacts with the database through calls to GDI functions.

Database Design

EvLoc uses a database for recording station- and network-magnitude results. The entity-relationship model of the schema used to estimate event magnitudes is indicated in Figure 12 on page 124.

Database Schema

Table 49 summarizes the usage of database tables by *EvLoc*. The first two columns identify the table and whether it is read or written, and the third column shows the purpose for reading or writing each attribute. Only those attributes that are required to obtain station- and network-magnitude estimates are included in this table. The relationships between the database tables themselves are shown in Figure 12.

TABLE 49: EVLOC DATABASE USAGE FOR MAGNITUDE ESTIMATION

Table	Action	Usage
affiliation	reads	• *net* for record identification • *sta* for linking *net* to **site** records
site	reads	• *sta* for record identification and linking stations with station-specific TLM description data stored in TLSF • *ondate* and *offdate* for record identification • *lat* and *lon* for determining event-to-station distance
origin	reads	• *orid*, *evid*, and *jdate* for record identification • *lat* and *lon* for determining event-to-station distance and retrieving magnitude correction from TLM • *depth* for identifying usable TLM and retrieving magnitude correction from TLM
assoc	reads	• *arid* and *orid* for record identification • *sta*, *phase*, and *delta* for identifying TLM to be used with arrival-based amplitude • *timedef* for optionally restricting which station magnitude may be magnitude-defining

TABLE 49: EVLOC DATABASE USAGE FOR MAGNITUDE ESTIMATION (CONTINUED)

Table	Action	Usage
parrival	reads	• *parid* and *evid* for record identification • *sta* and *phase* for identifying TLM to be used with origin-based amplitude
amplitude	reads	• *ampid*, *arid*, *parid*, and *amptype* for record identification • *chan* for identifying usable TLM • *amp*, *per*, and *duration* for computing station magnitude • *clip* for flagging clipped amplitude
stamag	reads	• *ampid*, *arid*, *orid*, and *magtype* for record identification • *sta* for determining event-station distance, identifying usable TLM, and identifying substation magnitudes • *phase* for identifying usable TLM • *magdef* for identifying defining or nondefining state of station magnitude • *auth* for optionally using pre-existing *magdef* state of station magnitude
netmag	reads	• *orid* and *magtype* for record identification
event_control	reads	• *orid* for record identification • *mag_sdv_screen* and *mag_sdv_mult* for requesting residual outlier screening • *mag_alpha_only* for using only a primary set of stations in computing magnitudes • *mb_min_dist* and *mb_max_dist* for defining minimum and maximum distance range for valid mb amplitude data
origin	writes	• *mbid*, *msid*, and *mlid* for record identification • *mb*, *ms*, and *ml* for recording network magnitudes • *auth* for recording application and user • *lddate* for recording creation date of new **origin** record

TABLE 49: EVLOC DATABASE USAGE FOR MAGNITUDE ESTIMATION (CONTINUED)

Table	Action	Usage
stamag	writes	• *magid*, *ampid*, *arid*, *orid*, *evid* for record identification; *magid* may be new • *sta* and *phase* for preserving station/phase pair • *delta* for recording event-to-station distance • *magtype* for recording magnitude type • *magnitude* and *uncertainty* for recording station magnitude and uncertainty • *magres* for recording final magnitude residual • magdef for recording whether or not the station magnitude was used to estimate network magnitude • *mmodel* for recording magnitude model designation • *auth* for recording application and user • *lddate* for recording creation data of new **stamag** record
netmag	writes	• *magid*, *orid*, and *evid* for record identification; *magid* may be new • *net* for recording network used to estimate network-magnitude data • *magtype* for recording magnitude type • *nsta* for recording number of magnitude-defining station magnitudes used to estimate network-magnitude data • *magnitude* and *uncertainty* for recording network magnitude and uncertainty • *auth* for recording application and user • *lddate* for recording creation data of new **netmag** record
event_control	writes	• *orid* and *evid* for record identification • *mag_sdv_screen* and *mag_sdv_mult* for indicating if residual outlier screening was applied • *mag_alpha_only* for indicating if only a primary set of stations was used to estimate magnitudes • *mb_min_dist* and *mb_max_dist* for recording minimum and maximum distance range used to limit mb amplitude data

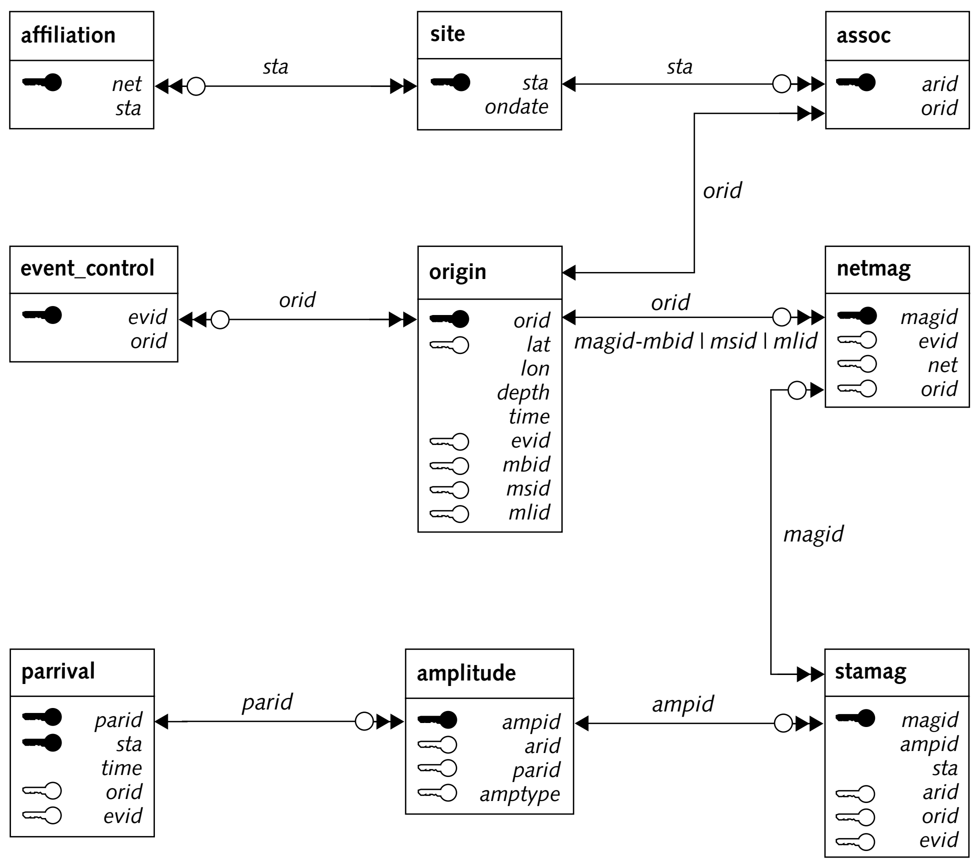


FIGURE 12. EVENT MAGNITUDE DATABASE TABLE RELATIONSHIPS

EvLoc Data Structures

EvLoc uses C data structures for storing station and event data and control parameters in internal memory. The following paragraphs describe the *EvLoc* data structures that are needed for magnitude processing.

EvLoc_Par

The `EvLoc_Par` structure contains general control-parameter settings. These settings are valid for a single *EvLoc* execution. They may be a combination of default settings and user-specific settings listed in an input parameter file. The general control parameters are specified by `read_evloc_par()`. This structure is a component of the control-parameter-data memory store (M3 in Figure 7 on page 32; Table 5 on page 33). Refer to the *EvLoc* man page for descriptions and default values of the general control parameters. The "M" column in Table 50 indicates whether or not the `EvLoc_Par` structure member is magnitude-related.

TABLE 50: EVLOC_PAR STRUCTURE

Type	Name	Description	M
`Bool`	`triple_location`	estimate event location and magnitude at three depths (free depth, surface, restrained depth)? `0` = no, `1` = yes	yes
`int`	`mode`	estimate event locations/magnitudes? `0` = event locations only, `1` = magnitudes only, `2` = both event locations and magnitudes	yes
`int`	`max_gdi_records`	maximum number of records read from or written to database	yes
`Bool`	`write_to_input_db_tables`	write output magnitude data to input database tables? `0` = no, `1` = yes	yes
`char *`	`input_db_account`	input database account	yes
`char *`	`output_db_account`	output database account	yes
`char *`	`db_vendor`	database vendor	yes
`char *`	`net`	unique network identifier	yes
`char *`	`affiliation_table`	name of input **affiliation** table	yes
`char *`	`aoi_file`[1]	area-of-interest file pathname	no
`char *`	`site_table`	name of input **site** table	yes
`char *`	`origin_table`	name of input **origin** table	yes

TABLE 50: EVLOC_PAR STRUCTURE (CONTINUED)

Type	Name	Description	M
char *	arrival_table	name of input **arrival** table	no
Bool	use_prev_magdefs	use magnitude-defining states of input (pre-existing) station-magnitude data? 0 = no, 1 = yes	yes
char *	det_amplitude_table	name of input **amplitude** table containing arrival-based amplitude data	yes
char *	ev_amplitude_table	name of input **amplitude** table containing origin-based amplitude data	yes
char *	parrival_table	name of input **parrival** table	yes
char *	netmag_table	name of input **netmag** table	yes
char *	stamag_table	name of input **stamag** table	yes
char *	assoc_table	name of input **assoc** table	yes
char *	event_control_table	name of input **event_control** table	yes
char *	origerr_table	name of input **origerr** table	no
char *	new_origin_table	name of output **origin** table	yes
char *	new_origerr_table	name of output **origerr** table	no
char *	new_assoc_table	name of output **assoc** table	no
char *	new_netmag_table	name of output **netmag** table	yes
char *	new_stamag_table	name of output **stamag** table	yes
char *	origin_query	database query used to retrieve data from input **origin** table	yes
Bool	use_ev_cntrl_table	override magnitude control parameters with data from input **event_control** table? 0 = no, 1 = yes	yes
Bool	write_ev_cntrl_table	write magnitude control parameters to output **event_control** table? 0 = no, 1 = yes	yes
Bool	write_ar_info_table	write association-based measurements to output **ar_info** table? 0 = no, 1 = yes	no

TABLE 50: EVLOC_PAR STRUCTURE (CONTINUED)

Type	Name	Description	M
char *	ar_info_table	name of output **ar_info** table	no
Bool	write_af_tables[1]	write database extension tables to output database account? 0 = no, 1 = yes	no
char *	af_origin_table[1]	name of output **origin** table	yes
char *	af_origerr_table[1]	name of output **origerr** table	no
char *	af_assoc_table[1]	name of output **assoc** table	no
char *	af_netmag_table[1]	name of output **netmag** table	yes
char *	af_stamag_table[1]	name of output **stamag** table	yes
Bool	create_syn_data_only	create synthetic arrival data? 0 = no, 1 = yes	no
Bool	add_gauss_noise_to_syn	add Gaussian noise to synthetic travel times, azimuths, and slowness? 0 = no, 1 = yes	no
char *	new_arrival_table	name of output **arrival** table	no
char **	phases	phase names used to determine event locations	no
int	num_phases[2]	number of phase names used to determine event locations	no
Bool	sub_sta_list_only	use only event data for stations listed in sub_sta_list to estimate event locations? 0 = no, 1 = yes	no
char **	sub_sta_list	stations within network used to estimate event locations	no
int	num_sub_sta_list[2]	number of stations within network used to estimate event locations	no
char **	list_of_magtypes	magnitude descriptors for which network-magnitude data are to be estimated	yes
int	num_magtypes[2]	number of magnitude descriptors for which network-magnitude data are to be estimated	yes

TABLE 50: EVLOC_PAR STRUCTURE (CONTINUED)

Type	Name	Description	M
char **	list_of_magtypes_ to_timedef_restrict	magnitude descriptors for which the magnitude-defining state may only be defining if the associated time-defining state is defining	yes
int	num_magtypes_to_ timedef_restrict[2]	number of magnitude descriptors for which the magnitude-defining state may only be defining if the associated time-defining state is defining	yes
char *	sasc_dir_prefix	directory pathname and filename prefix for slowness/azimuth station correction tables	no
char *	mag_descrip_file	MDF pathname	yes
char *	tl_spec_file	TLSF pathname	yes

1. This member is not applicable to the IDC.

2. This member is defined during `read_evloc_par()` processing and does not have an associated parameter file argument.

Mag_Params

The `Mag_Params` structure contains general magnitude control parameter settings. These settings are valid for a single *EvLoc* execution. They may be a combination of default settings and user-specific settings listed in an input parameter file. The general magnitude control parameters are specified by `read_evloc_par()`. This structure is a component of the control-parameter-data memory store (*M*3 in Figure 7 on page 32; Table 5) and the event-data memory store (*M*5 in Figure 7; Table 7). Some members of this structure in *M*5 may be updated by the magnitude control parameters read from the input **event_control** tables on an event-by-event basis. Refer to the *EvLoc* man page for descriptions and default values of the general magnitude control parameters.

TABLE 51: MAG_PARAMS STRUCTURE

Type	Name	Description
int	verbose	level of verbosity for printed magnitude output
char[9]	net	unique network identifier
char[7]	magtype_to_origin_mb	m_b magtype for which network-magnitude data are written to output **origin**.*mb* and *mbid* fields
char[7]	magtype_to_origin_ms	M_s magtype for which network-magnitude data are written to output **origin**.*ms* and *msid* fields
char[7]	magtype_to_origin_ml	ML magtype for which network-magnitude data are written to output **origin**.*ml* and *mlid* fields
char **	list_of_mb_magtypes	m_b magtypes for which magnitude control data are retrieved from input **event_control** table
int	num_mb_magtypes[1]	number of m_b magtypes for which magnitude control data are retrieved from input **event_control** table
int	num_boots	maximum number of bootstrap resamples permitted
Bool	use_only_sta_w_corr[2]	use only amplitude data from stations with test-site magnitude corrections? 0 = no, 1 = yes
Bool	sub_sta_list_only	use only event data for stations listed in sub_sta_list to estimate magnitude data? 0 = no, 1 = yes
char **	sub_sta_list	stations within network used to estimate magnitude data
int	num_sub_sta_list[1]	number of stations within network used to estimate magnitude data
Bool	ignore_large_res	ignore station-magnitude data with large station-magnitude residuals? 0 = no, 1 = yes
double	large_res_mult	station-magnitude residual multiplication scale factor
Bool	use_ts_corr[2]	apply test-site magnitude corrections? 0 = no, 1 = yes

TABLE 51: MAG_PARAMS STRUCTURE (CONTINUED)

Type	Name	Description
`char[9]`	`ts_region`[2]	test-site magnitude region descriptor
`char *`	`outfile_name`	output pathname to which output magnitude data should be written

1. This variable is defined during `read_evloc_par()` processing, and does not have an associated parameter file argument.
2. This member is not applicable to the IDC.

Ev

The `Ev` linked list contains input event data and output event location and magnitude data for a single event. The input event data are read from the input database and stored in this linked list by `read_evloc_db_tables()`. The output event location and magnitude data are populated by *libloc* and *libmagnitude* processing units, and are copied from this linked list to the output database by `write_evloc_db_tables()`. This linked list is a component of the event-data memory store (M5 in Figure 7 on page 32; Table 7 on page 34).

TABLE 52: EV LINKED LIST

Type	Name	Description
`int`	`prefer_loc`[1]	preferred location identifier
`Bool[3]`	`write_this_solution`[1]	write event location and magnitude of origin to output database tables? `0` = no, `1` = yes
`Event_control *`	`event_control`	pointer to (array of)[2] `Event_control` database table structure(s)
`Origin *`	`origin`	pointer to (array of)[2] `Origin` database table structure(s)
`Arrival *`	`arrival`[1]	pointer to array of `Arrival` database table structures

TABLE 52: EV LINKED LIST (CONTINUED)

Type	Name	Description
`Assoc **`	`assoc`	array of pointers to `Assoc` database table structures
`Ar_Info **`	`ar_info`[1]	array of pointers to `Ar_Info` database table structures
`int`	`num_assocs`	number of elements in array of `Assoc` structures
`Loc_ptr *`	`loc`[1]	pointer to (array of)[2] `Loc_ptr` structures
`Locator_params *`	`loc_params`[1]	pointer to `Locator_params` structure
`Mag_ptr *`	`mag`	pointer to (array of)[2] `Mag_ptr` structure(s)
`Mag_Params *`	`mag_params`	pointer to `Mag_Params` structure
`Ev *`	`next`	pointer to next element in `Ev` linked list

1. Not used if only magnitudes are being determined. Refer to [IDC-7.1.5] for a description of these linked list members.
2. This member is a pointer to an array of structures if more than one origin is being located (and magnitudes determined) for the event.

Mag_Ptr

The `Mag_Ptr` structure contains event (primarily magnitude) data for a single event. The most important structure member is the array of `Magnitude` objects (Table 54). Each element of the array corresponds to a different event magtype. The event data are stored in this structure by `read_evloc_db_tables()`. This structure is a component of the event-data memory store (M5 in Figure 7 on page 32; Table 7 on page 34).

TABLE 53: MAG_PTR STRUCTURE

Type	Name	Description
int	num_mags	number of elements in array of Magnitude objects
Magnitude *	magnitude	array of Magnitude objects
Af_netmag *	af_netmag[1]	array of Af_netmag database table structure
Af_stamag **	af_stamag[1]	array of pointers to Af_stamag database table structures

1. This member is not applicable to the IDC.

libmagnitude Data Structures

libmagnitude uses C data structures for storing event data, magnitude control parameters, and earth-model data in internal memory. The following paragraphs describe the most important *libmagnitude* data structures.

Magnitude

The `Magnitude` object contains event and magnitude specification data for a single event and magtype. The event data are primarily records read from an input database and stored in database table structures by a calling application. The magnitude specification data are magnitude control settings that are originally defined in the magnitude description data section of the MDF (see Table 55), but may be modified by the application. A `Magnitude` object provides a convenient way to bind amplitude and magnitude data together by magtype and pass these data between an application and *libmagnitude* processing units. `build_mag_obj()` and the lower-level functions it calls store the event and magnitude specification data in the `Magnitude` object. This structure is a component of the magnitude-data memory store (M2 in Figure 10; Table 11 on page 45).

TABLE 54: MAGNITUDE OBJECT

Type	Name	Description
Bool	mag_computed	successful network magnitude estimated? FALSE = no, TRUE = yes
Bool	mag_write	write Stamag and Netmag database table structure contents to output database tables? FALSE = no, TRUE = yes
Mag_Cntrl	mag_cntrl	Mag_Cntrl structure
Netmag	netmag	Netmag database table structure
Stamag **	stamag	array of pointers to Stamag database table structures
Amplitude **	amplitude	array of pointers to Amplitude database table structures
SM_Aux *	sm_aux	array of SM_Aux structures
int	count	number of elements in Stamag and Amplitude structures

Mag_Descrip

The Mag_Descrip structure contains magnitude description data (control settings) for a single magtype. The control settings define what event data will be used to estimate station magnitudes, how a network magnitude will be estimated from station magnitudes, and how the network-magnitude standard deviation may be bounded. The magnitude description data are read from the MDF by read_ mdf(). This structure is a component of the earth-model-data memory store (M1 in Figure 8 on page 37, Figure 9 on page 41, and Figure 10 on page 43; Table 8 on page 38). The Mag_Cntrl structure, which contains magnitude specification data, is composed of the first 11 members of the Mag_Descrip structure.

TABLE 55: MAG_DESCRIP STRUCTURE

Type	Name	Description
char[7]	magtype	magnitude descriptor
char[9]	TLtype	transmission-loss descriptor
char[9]	det_amptype	amplitude measure descriptor for arrival-based amplitudes
char[9]	ev_amptype	amplitude measure descriptor for origin-based amplitudes
int	algo_code	magnitude algorithm code: 0 = network average, 1 = MLE without bootstrapping, 2 = MLE with bootstrapping
float	dist_min	minimum valid distance (deg)
float	dist_max	maximum valid distance (deg)
float	sglim1	lower-bound standard deviation
float	sglim2	upper-bound standard deviation
float	sgbase	baseline standard deviation
Bool	apply_wgt	estimate weighted average magnitudes? 0 = no, 1 = yes
float	def_sta_corr	default bulk station correction
float	def_sta_corr_err	default bulk-station-correction error
char[9]	orig_det_amptype	original amplitude measure descriptor for arrival-based amplitudes
char[9]	orig_ev_amptype	original amplitude measure descriptor for origin-based amplitudes
int	orig_algo_code	original magnitude algorithm code
float	orig_dist_min	original minimum valid distance (deg)
float	orig_dist_max	original maximum valid distance (deg)
float	orig_sglim1	original lower-bound uncertainty
float	orig_sglim2	original upper-bound uncertainty

TABLE 55: MAG_DESCRIP STRUCTURE (CONTINUED)

Type	Name	Description
float	orig_sgbase	original baseline uncertainty
Bool	orig_apply_wgt	original setting for computing weighted average magnitudes

Mag_Sta_TLType

The `Mag_Sta_TLType` structure contains bulk-station-correction data (static station-magnitude corrections and errors) for a single station and TLtype combination. The bulk-station-correction data are read from the MDF by `read_mdf()`. This structure is a component of the earth-model-data memory store (M1 in Figures 8, 9, and 10; Table 8 on page 38).

TABLE 56: MAG_STA_TLTYPE STRUCTURE

Type	Name	Description
char[7]	sta	station name
char[9]	TLtype	transmission-loss descriptor
float	bulk_sta_corr	bulk station correction
float	bulk_sta_corr_err	bulk-station-correction error

SM_Aux

The `SM_Aux` structure contains auxiliary station-magnitude data for a single amplitude and magtype. `build_mag_obj()` determines and stores the auxiliary station-magnitude data. This structure is nested within the `Magnitude` object (Table 54) as a component of the magnitude-data memory store (M2 in Figure 10; Table 11 on page 45).

TABLE 57: SM_AUX STRUCTURE

Type	Name	Description
Bool	detect_based	arrival-based amplitude? 0 = no, 1 = yes
Bool	manual_override	retain magnitude-defining state of associated station magnitude throughout network-magnitude processing? 0 = no, 1 = yes
Bool	clipped	clipped amplitude? 0 = no, 1 = yes
int	sig_type	signal type: 1 = arrival-based amplitude, 2 = origin-based amplitude, 3 = clipped amplitude
double	wt	station-magnitude uncertainty

SM_Info

The SM_Info structure contains station-magnitude data for a single amplitude and magtype. The station-magnitude data are estimated by station_magnitude() and its lower-level functions. If the calling application operates in station-magnitude mode, then this structure composes the station-magnitude-data memory store (M7 in Figure 8 on page 37; Table 9 on page 40). If the application operates in network-magnitude mode, then this structure is contained within the *Estimate Station-magnitude Data* process (2.3 in Figure 10) and is therefore not shown as a component of a memory store.

TABLE 58: SM_INFO STRUCTURE

Type	Name	Description
int	mag_error_code	error code returned from interpolating transmission-loss model
double	sta_magnitude	station magnitude
int	src_dpnt_corr_type[1]	test-site magnitude correction requested? 0 = no, 1 = yes

TABLE 58: SM_INFO STRUCTURE (CONTINUED)

Type	Name	Description
double	total_mag_corr	total magnitude correction computed as sum of mc_table_value and bulk_static_sta_corr
double	mc_table_value	distance/dependent magnitude correction
double	bulk_static_sta_corr	bulk station-magnitude correction
double	bulk_sta_corr_error	bulk station-magnitude correction error
double	src_dpnt_corr[1]	test-site magnitude correction
double	model_error	modeling error
double	meas_error	measurement error
double	model_plus_meas_error	total magnitude uncertainty computed as rms of the model_error, meas_error, and bulk_sta_corr_error
double[4]	mag_cor_deriv	first and second derivatives of transmission loss with respect to distance and depth; the first derivatives are stored in first and second elements of array, and second derivatives are stored in third and fourth elements of array
char[16]	mmodel	transmission-loss model
char[18]	lddate	load date

1. This member is not applicable to the IDC.

SM_Sub

The SM_Sub structure contains a subset of the station-magnitude data for a single station magnitude. The data subset are stored in this structure by calc_mags() and are only accessed by other functions within the *Estimate Network-magnitude Data* process. This structure is a component of the magnitude-data memory store (M2 in Figure 10 on page 43 and Figure 11 on page 48; Table 11 on page 45).

TABLE 59: SM_SUB STRUCTURE

Type	Name	Description
char[2]	magdef	"d" or "n" flag indicating defining or nondefining state of station magnitude
int	sig_type	signal type: 1 = arrival-based amplitude, 2 = origin-based amplitude, 3 = clipped amplitude
double	wt	station-magnitude uncertainty
double	magnitude	station magnitude

Sta_TL_Model

The `Sta_TL_Model` structure contains optional station-specific TLM description data for a single station and TLtype combination. The station-specific TLM description data list the root name, optional phase name, and optional channel/frequency identifier associated with a particular station and TLtype combination. This information is used in conjunction with TLM pathway data from the `TL_Model_Path` structures (Table 63) to create a pathname that points to a unique, station-specific TLM. The magnitude corrections and modeling errors within this TLM are to be used to estimate a station magnitude for a given station and TLtype pair. The TLtype is linked to a magtype through the `Mag_Descrip` structure (Table 55). The station-specific TLM description data are read from the TLSF by `read_tlsf()`. This structure is a component of the earth-model-data memory store (M1 in Figure 8 on page 37, Figure 9 on page 41, and Figure 10 on page 43; Table 8 on page 38).

TABLE 60: STA_TL_MODEL STRUCTURE

Member Name	Data Type	Description
sta	char[7]	station identifier
TLtype	char[9]	transmission-loss descriptor

TABLE 60: STA_TL_MODEL STRUCTURE (CONTINUED)

Member Name	Data Type	Description
model	char[16]	station-specific TLM root name
phase	char[9]	phase name
chan	char[9]	channel identifier
tl_index	int	index of element in TL_Pt linked list that has identical TLtype
model_index	int	index of element in array of TL_Model_Path structures that has identical model member

TLType_Model_Descrip

The TLType_Model_Descrip structure contains default TLM description data for a single TLtype. The default TLM description data list the root name and the phase names associated with a single TLtype. This information is used in conjunction with TLM pathway data from the TL_Model_Path structures (Table 63) to create a pathname that points to a default TLM for a TLtype. The magnitude corrections and modeling errors within this TLM are to be used to estimate station magnitudes for the magtype associated with the TLtype in the Mag_Descrip structures (Table 55). The default TLM description data are read from the TLSF by read_tlsf(). This structure is a component of the earth-model-data memory store (M1 in Figure 8 on page 37, Figure 9 on page 41, and Figure 10 on page 43; Table 8 on page 38).

TABLE 61: TLTYPE_MODEL_DESCRIP STRUCTURE

Type	Name	Description
char[9]	TLtype	transmission-loss descriptor
char[16]	model	default TLM root name
int	model_index	index of element in array of TL_Model_Path structures that has identical model member

TABLE 61: TLTYPE_MODEL_DESCRIP STRUCTURE (CONTINUED)

Type	Name	Description
Bool	phase_dependency	default TLM phase-dependent? 0 = no, 1 = yes
List_of_Phz *	list_of_phz	pointer to List_of_Phz linked list

TL_Mdl_Err

The TL_Mdl_Err structure contains transmission-loss modeling-error data read from a single TLM. The modeling errors (that is, standard deviations) are estimates of the transmission-loss values. They may be distance/depth-dependent, distance-dependent only, or may be condensed into a single, global value representing the modeling error for the entire TLM. The transmission-loss modeling error data are read from the TLM by read_tl_table(). This structure is nested within the TL_Table structure (Table 64) as a component of the earth-model-data memory store (M1 in Figure 8 on page 37, Figure 9 on page 41, and Figure 10 on page 43; Table 8 on page 38).

TABLE 62: TL_MDL_ERR STRUCTURE

Type	Name	Description
float	bulk_var	single, global transmission-loss modeling error
int	num_dists	number of distance samples
int	num_depths	number of depth samples
float *	dist_samples	array of distance samples for which transmission-loss modeling errors are valid (deg)
float *	depth_samples	array of depth samples for which transmission-loss modeling errors are valid (deg)
float *	dist_var	array of distance-dependent transmission-loss modeling errors
float **	dist_depth_var	two-dimensional array of distance/depth-dependent transmission-loss modeling errors

TL_Model_Path

The `TL_Model_Path` structure contains the root name of a TLM and its pathway relative to the directory location of the TLSF. The TLM pathway data are read from the TLSF by `read_tlsf()`. This structure is a component of the earth-model-data memory store (M1 in Figure 8 on page 37, Figure 9 on page 41, and Figure 10 on page 43; Table 8 on page 38).

TABLE 63: TL_MODEL_PATH STRUCTURE

Type	Name	Description
`char[16]`	`model`	default TLM root name
`char *`	`dir_pathway`	directory location of TLM relative to TLSF

TL_Table

The `TL_Table` structure contains transmission-loss (magnitude correction) data and transmission-loss modeling error data read from a single TLM. The transmission-loss data are distance/depth-dependent estimates of the transmission loss incurred as a signal propagates from an event to a station. The modeling error data are estimates of the modeling errors in the transmission-loss values. The transmission-loss and modeling error data are read from a single TLM by `read_tl_table()`. This structure is a component of the earth-model-data memory store (M1 in Figure 8, Figure 9, and Figure 10; Table 8).

TABLE 64: TL_TABLE STRUCTURE

Type	Name	Description
`char[9]`	`TLtype`	transmission-loss descriptor
`char[16]`	`model`	station-specific TLM root name
`char[9]`	`phase`	phase name
`char[9]`	`chan`	channel identifier
`int`	`num_dists`	number of distance samples

TABLE 64: TL_TABLE STRUCTURE (CONTINUED)

Type	Name	Description
int	num_depths	number of depth samples
float[2]	in_hole_dist	minimum and maximum distance encompassing any holes in the TLM (deg)
float *	dist_samples	array of distance samples for which transmission-loss values are estimated (deg)
float *	depth_samples	array of depth samples for which transmission-loss values are estimated (deg)
float **	tl	two-dimensional array of transmission-loss values (that is, magnitude corrections)
TL_Mdl_Err *	tl_mdl_err	pointer to TL_Mdl_Err structure
int	num_ts_regions[1]	number of elements in TL_TS_Cor structure.
TL_TS_Cor *	tl_ts_cor1	pointer to TL_TS_Cor structure

1. This member is not applicable to the IDC.

References

The following sources supplement or are referenced in document:

[Bla82] Blandford, R. R. and R. H. Shumway, "Magnitude: Yield for Nuclear Explosions in Granite at the Nevada Test Site and Algeria: Joint Determination with Station Effects and with Data Containing Clipped and Low-Amplitude Signals," *Seismic Data Analysis Center, Teledyne Geotech*, VSC-TR-82-12, 1982.

[DOD94a] Department of Defense, "Software Design Description," *Military Standard Software Development and Documentation,* MIL-STD-498, 1994.

[Gan79] Gane, C., and Sarson, T., *Structured Systems Analysis: Tools and Techniques*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1979.

[IDC5.1.1Rev2] Science Applications International Corporation, Veridian Pacific-Sierra Research, *Database Schema, Revision 2*, SAIC-00/3057, PSR-00/TN2830, 2000.

[IDC5.2.1] Science Applications International Corporation, *IDC Processing of Seismic, Hydroacoustic, and Infrasonic Data*, SAIC-99/3023, 1999.

[IDC6.5.1] Science Applications International Corporation, *Interactive Analysis Subsystem Software User Manual,* SAIC-01/3001, 2001.

[IDC6.5.2Rev0.1] Science Applications International Corporation, *Distributed Application Control System (DACS) Software User Manual, Revision 0.1*, SAIC-00/3038, 2000.

[IDC7.1.1] Science Applications International Corporation, *Detection and Feature Extraction (DFX)–Scheme Files*, SAIC-01/3000, 2001.

[IDC7.1.5] Science Applications International Corporation, *Event Location Software,* SAIC-01/3010, 2001.

[McL88] McLaughlin, K., "Maximum Likelihood Event Magnitude Estimation with Bootstrapping for Uncertainty Estimation," *Bulletin of the Seismological Society of America*, Volume 78, pp. 855–862, 1988.

Glossary

A

amplitude

Zero-to-peak height of a waveform in nanometers.

amptype

Descriptor that uniquely identifies an amplitude measurement type (for example, `A5/2` or `SBSNR`).

analyst

Personnel responsible for reviewing and revising the results of automatic processing.

Analyst Review Station

This application provides tools for a human analyst to refine and improve the event bulletin by interactive analysis.

arrival

Detected signal that has been associated to an event. First, the Global Association (GA) software associates the detection to an event. Later, during interactive processing, many arrivals are confirmed, improved, or added by visual inspection.

arrival-based amplitude

Amplitude measured by *DFX* for a detected signal.

ARS

See Analyst Review Station.

ASCII

American Standard Code for Information Interchange. Standard, unformatted 256-character set of letters and numbers.

attribute

(1) Database column. (2) Characteristic of an item; specifically, a quantitative measure of a S/H/I detection such as azimuth, slowness, period, or amplitude.

azimuth

Direction, in degrees clockwise with respect to North, from a station to an event.

B

bootstrap resampling

Statistical technique of random resampling of data elements with replacement (that is, without regard to which elements have already been selected) used to estimate errors in parameters estimated from a distribution.

bulk station correction

Empirical station- and TLtype-specific term added to the logarithm of the amplitude during computation of a station magnitude to correct for local station effects.

bulk station correction error

Estimate of the standard error associated with the bulk station correction.

C

channel

Component of motion or distinct stream of data.

CMR

Center for Monitoring Research.

command

Expression that can be input to a computer system to initiate an action or affect the execution of a computer program.

commit

Process of saving changes made to the database.

component

(1) One dimension of a three-dimensional signal; (2) The vertically or horizontally oriented (north or east) sensor of a station used to measure the dimension; (3) One of the parts of a system; also referred to as a module or unit.

Comprehensive Nuclear-Test-Ban Treaty Organization

Treaty User group that consists of the Conference of States Parties, the Executive Council, and the Technical Secretariat.

computer software component

Functionally or logically distinct part of a computer software configuration item; possibly an aggregate of two or more software units.

computer software configuration item

Aggregation of software that is designated for configuration management and treated as a single entity in the configuration management process.

configuration

(1) (hardware) Arrangement of a computer system or components as defined by the number, nature, and interconnection of its parts. (2) (software) Set of adjustable parameters, usually stored in files, which control the behavior of applications at run time.

connection

Open communication path between protocol peers.

COTS

Commercial-Off-the-Shelf; terminology that designates products such as hardware or software that can be acquired from existing inventory and used without modification.

CSC

See computer software component.

CSCI

See computer software configuration item.

CTBTO

See Comprehensive Nuclear-Test-Ban Treaty Organization.

D

DACS

See Distributed Application Control System.

database table structure

C structure that is structurally equivalent to the schema of a database table.

defining magnitude

Station magnitude that is used to compute a network magnitude.

deg.

Degrees (as a distance).

detection

Probable signal that has been automatically detected by the Detection and Feature Extraction (*DFX*) software.

Detection and Feature Extraction

DFX is a programming environment that executes applications written in Scheme (known as *DFX* applications).

DFX

See Detection and Feature Extraction.

Distributed Application Control System

This software supports inter-application message passing and process management.

E

element

Single station or substation of a sensor array, referred to by its element name (such as YKR8), as opposed to its array name (YKA in this example). (2) Data storage location in a data array.

event

Unique source of seismic, hydroacoustic, or infrasonic wave energy that is limited in both time and space.

EvLoc

Application used to compute event location and/or magnitude.

execute

Carry out an instruction, process, or computer program.

external interface

Library processing unit that exchanges data with an application.

external memory store

Memory store accessed by multiple applications or libraries.

F

field

(1) Attribute of a generic object.
(2) Attribute in a database table (the name of the column).

filesystem

Named structure containing files in subdirectories. For example, UNIX can support many filesystems; each has a unique name and can be attached (or mounted) anywhere in the existing file structure.

function

Named section of a program that performs a particular task.

G

GA

See Global Association.

GAcons

GA application that precomputes propagation knowledge base information and stores it in two grid files used by GA.

GB

Gigabyte. A measure of computer memory or disk space that is equal to 1,024 megabytes.

GDI

Generic Database Interface.

Global Association

Subsystem that associates S/H/I phases to events.

grid

Set of points used by GA covering either a region of the earth or the whole earth and including the interior where deep seismicity occurs. Information about propagation to a network of stations is computed by *GAcons* for a grid and stored in a binary file.

GSETT-3

Group of Scientific Experts Third Technical Test.

I

IDC

International Data Centre.

IMS

International Monitoring System.

internal interface

Library processing unit that exchanges data only with other processing units in the same library.

internal memory store

Memory store accessed only by a single application or library.

IPC

Interprocess communication. The messaging system by which applications communicate with each other through *libipc* common library functions. See *tuxshell*.

K

km

Kilometer.

L

LAN

Local Area Network.

libgdi

Library containing functions for RDBMS access.

linked list

List of similar data structures linked to one another through the use of pointers. A linked list can be uni-directional (the pointer is always to the next element in the list) or bi-directional (there are pointers to both the previous and next elements in the list).

lower-bound magnitude

Arithmetic mean of a set of station magnitudes that were computed exclusively from clipped amplitudes.

M

magnitude

Empirical measure of the size of an event (usually made on a log scale).

magnitude correction

A correction added to the logarithm of the amplitude during computation of a station magnitude.

magnitude correction table

ASCII file representation of a Transmission Loss Model.

Magnitude Description File

File that maps amptypes and TLtypes to magtypes and specifies magnitude control settings and bulk station correction data.

magnitude-defining

See defining magnitude.

magtype

Descriptor that uniquely identifies a computed magnitude type (for example, `mb_ave` or `mb_mle`).

MB

Megabyte. 1,024 kilobytes.

m_b

Magnitude estimated from seismic body waves.

MDF

See Magnitude Description File.

member

A variable in a data structure.

M_L

Magnitude estimated from seismic waves measured near the source.

MLE

Maximum Likelihood Estimate.

modelling error (magnitude)

Estimate of the standard error associated with the transmission loss model correction.

M_s

Magnitude of seismic surface waves.

N

N/A

Not Applicable.

network

Spatially distributed collection of seismic, hydroacoustic, or infrasonic stations for which the station spacing is much larger than a wavelength.

network processing

Processing that uses the results of Station Processing from a network of stations to define and locate events.

network-average magnitude

Arithmetic mean of a set of station magnitudes computed from arrival-based amplitudes.

nm

Nanometer.

noise

Incoherent natural or artificial perturbations of the waveform trace caused by ice, animals migrations, cultural activity, equipment malfunctions or interruption of satellite communication, or ambient background movements.

NoiseAmp

Automatic Noise Amplitude Estimation. A *DFX* Scheme application that measures the noise level at stations that did not detect signals from a given event.

nondefining magnitude

Station magnitude that is not used to compute a network magnitude.

NULL

Empty, zero.

O

ORACLE

Vendor of the database management system used at the PIDC and IDC.

orid

Origin Identifier.

origin

Hypothesized time and location of a seismic, hydroacoustic, or infrasonic event. Any event may have many origins. Characteristics such as magnitudes and error estimates may be associated with an origin.

origin-based amplitude

Amplitude measured in a time window computed from the predicted travel time from the origin.

P

par

See parameter.

parameter

User-specified token that controls some aspect of an application (for example, database name, threshold value). Most parameters are specified using [*token* = *value*] strings, for example, `dbname=mydata/base@oracle`.

parameter (par) file

ASCII file containing values for parameters of a program. Par files are used to replace command line arguments. The files are formatted as a list of [*token* = *value*] strings.

parrival

Database table that contains the predicted arrivals and associations for origin-based amplitude measurements.

parse

Decompose information contained in a set of data.

pathname

Filesystem specification for a file's location.

period

Average duration of one cycle of a phase, in seconds per cycle.

phase

Arrival that is identified based on its path through the earth.

phase name

Name assigned to a seismic, hydroacoustic or infrasonic arrival associated with a travel path.

PIDC

Prototype International Data Centre.

pipeline

1) Flow of data at the IDC from the receipt of communications to the final automated processed data before analyst review. 2) Sequence of IDC processes controlled by the DACS that either produce a specific product (such as a Standard Event List) or perform a general task (such as station processing).

post-location processing

Software that computes various magnitude estimates and selects data to be retrieved from auxiliary stations.

pre-existing magnitude records

Input station and/or network magnitude data created by an earlier *EvLoc* or *ARS* run.

process

Function or set of functions in an application that perform a task.

processing unit

Software component of a larger entity such as a program.

program

Organized list of instructions that, when executed, causes the computer to behave in a predetermined manner. A program contains a list of variables and a list of statements that tell the computer what to do with the variables.

Q

query

Request for specific data from a database.

R

RAM

Random Access Memory.

RDBMS

Relational Database Management System.

REB

See Reviewed Event Bulletin.

regional

(1) (distance) Source-to-seismometer separations between a few degrees and 20 degrees. (2) (event) Recorded at distances where the first P and S waves from shallow events have traveled along paths through the uppermost mantle.

residual

Difference between the observed value for an attribute (for example, time, azimuth, slowness, or magnitude) and its corresponding theoretical value.

Reviewed Event Bulletin

Bulletin formed of all S/H/I events that have passed analyst inspection and quality assurance review.

rms

Root mean square.

root name

Base name in a filename, as distinguished from the path or suffix (for example, `qfvc` is the root name in the filename `qfvc.mb`).

S

s

Second(s) (time).

S/H/I

Seismic, hydroacoustic, and infrasonic.

SAIC

Science Applications International Corporation.

SASC

Slowness-Azimuth Station Corrections.

save

Store an analyzed event to the final database, thereby preventing further changes to the event.

schema

Database structure description.

seismic

Pertaining to elastic waves traveling through the earth.

slowness

Inverse of velocity, in seconds/degree; a large slowness has a low velocity.

snr

Signal-to-noise ratio.

Solaris

Name of the operating system used on Sun Microsystems hardware.

StaPro

Station Processing application for S/H/I data.

States Parties

Treaty user group who will operate their own or cooperative facilities, which may be National Data Centres.

station

Collection of one or more monitoring instruments. Stations can have either one sensor location (for example, BGCA) or a spatially distributed array of sensors (for example, ASAR).

station code (or ID)

(1) Code used to identify distinct stations. (2) Site code.

station processing

Processing based on data from a single station.

station weight

Weight used when combining station values into a network value. The weight is the inverse square of the station uncertainty.

status code

Integer code returned from a function to a calling function indicating whether or not it encountered any warning or error conditions.

structure

Software construct that collects one or more variables, possibly of different types, together under a single name for convenient handling.

T

theoretical arrival

Point where an arrival is expected to appear on a waveform, based on an event's location and depth.

TLM

See Transmission Loss Model.

TLSF

See Transmission Loss Specification File.

TLtype

Descriptor that uniquely identifies the transmission-loss model associated with a particular magnitude type (for example, `mb` or `ms`).

transmission loss correction

Empirical distance/depth-dependent correction added to the logarithm of the amplitude during computation of a station magnitude.

Transmission Loss Model

Distance- and depth-dependent magnitude corrections and modelling errors for an attenuation curve associated with a particular TLtype and optional phase type and channel/frequency identifier.

Transmission Loss Specification File

File that specifies all mappings between global and regional transmission loss types and models.

tuxshell

Process in the Distributed Processing CSCI used to execute and manage applications. See IPC.

U

uncertainty

Estimate of the deviation from the true mean for the parameter or variable of interest.

UNIX

Trade name of the operating system used by the Sun workstations.

upper-bound magnitude

Arithmetic mean of a set of station magnitudes computed exclusively from origin-based amplitudes.

W

WaveExpert

Application in the Automatic Processing CSCI that determines data intervals to request from auxiliary stations.

waveform

Time-domain signal data from a sensor (the voltage output) where the voltage has been converted to a digital count (which is monotonic with the amplitude of the stimulus to which the sensor responds).

weighted-average magnitude

Network magnitude that is estimated by weighting each defining station magnitude by its station weight.

workstation

High-end, powerful desktop computer preferred for graphics and usually networked.

Index

F

G

H

I

L

M

N

O

P

R

S

T

W